MASTER IN COMPUTER SCIENCE

Migrating from single software development to a software product line application to the film critik website

Michel, Raphaël

Award date:
2009

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'Informatique
Année académique 2008-2009

Migrating from Single Software Development to a Software Product Line : Application to the Film Critik website

Raphaël Michel

Mémoire présenté en vue de l'obtention du grade de licencié en informatique

Migrating from Single Software Development to a Software Product Line : Application to Film Critik Website

Raphaël Michel

August 31, 2009

Abstract

Software product lines translate the concepts of industrial product lines to the world of software development in order to take the same types of advantages, applied to software products. Among the more cited, reduced time to market, increased quality, mass customization, and reduced development cost are some of the advantages brought when adopting a software product line approach.

Using a family of similar websites as example, this work presents importants concepts and steps to take to build a simple software product line. The whole family is analyzed to determine what commonalities and differences exist between the different websites in order to build reusable components that will be assembled together to form end-products.

Keywords : software product line, planned reuse, reusable component, website

Les lignes de produits logiciels transposent les concepts des lignes de produits industriels dans le monde du développement de logiciels afin de de bénéficier des mêmes types d'avantages, appliqués aux produits logiciels. Parmi les plus cités, la réduction du temps nécessaire pour mettre un produit sur le marché, l'amélioration de la qualité, la personnalisation de masse et la réduction du temps de développement sont quelques uns des avantages obtenus lors de l'adoption d'une ligne de produits logiciels.

En utilisant une famille de sites internet comme exemple, ce travail présente les concepts importants et les étapes à suivre pour mettre en place une ligne de produits logiciels simple. La famille entière est analysée pour déterminer quelles similarités et quelles différences existent entre les produits afin de pouvoir construire des composants réutilisables qui seront assemblés pour former des produits finis.

Mots-clés : ligne de produits logiciels, réutilisation planifiée, composant réutilisable, site internet

Acknowledgements

First of all, I would like to thank Mr Patrick Heymans, my supervisor, for his guidance, his advice and his patience, without him, this work would not have been possible. I'd also like to thank his assistants, who took the time to send me a lot of documentation that was very useful when redacting this document, the professors who taught me everything I knew before starting this as well as the rest of the staff of the FUNDP who helped me in various ways.

Besides them, a lot of other people contributed indirectly to this work by supporting me during the time it took to complete this work : my family, my parents, Lucien and Pascale, my brothers, Jérôme, Jérémy and his girlfriend Sarah, my fellow students, and especially Thibault, Piero and Cedric, my colleagues as well as my friends who always had kind words to encourage me.

To you all, and to all the people I have not mentioned here that showed interest in my work, thank you ! Without you, this work would not be what it is today.

Contents

Glossary

The ModX architecture uses some specific vocabulary. Here we try to define the most important terms.

Snippets :

Snippets or code snippets are blocks of PHP code that will be interpreted and executed.

Chunks :

Chunks are simple common text blocks, which can contain raw text, HTML code or anything else. The content a chunk is never interpreted and executed whatever it may be, but it can contain placeholders that can be replaced either by the template engine or directly by a snippet.

Placeholder :

Placeholders are character strings that are to be replaced by some real value at runtime. Their purpose is to tell the template engine where to put which values. Placeholders can be any character strings but ModX defines specific ways to form placeholders that tell the template engine where to get the value that needs to be displayed. The most common specific types of placeholders are calls to chunks and snippets.

Templates :

Templates are models for the HTML pages that will be generated and sent back to the visitor. The varying parts of the page are represented by placeholders.

Snippets :

Snippets or code snippets are blocks of PHP code that will be interpreted and executed.

CMS :

CMS stands for "Content Management System". CMS' offer an interface to manage the content of the website easily using various forms and graphical representations of the website hierarchy.

CMF :

CMF stands for "Content Management Framework". A CMF nearly always comes along with a CMS. It implements objects and APIs to allow developers to build new features easily in addition to everything that is already provided. The framework also allows to manage the content programatically.

Chapter 1

Introduction

1.1 History

Film-Critik.net was launched a few years ago. The main goal of the website has always been to post movie reviews that are short enough to be read in less than a minute, along with a few pieces of information about the movie.

The first version was entirely built from scratch in a short time. As one might guess, bugs were discovered and features had to be added or modified frequently. Most of the changes were often applied as "quick and dirty" fixes here and there and the code behind the website rapidly evolved in something that was becoming nearly impossible to maintain.

The second version brought some new features to the end user but its very aim was to restructure the code base to ensure easier maintainability and extensibility.

Figure 1.1: Screenshot of film-critik.net

1.2 Objectives of the thesis

The third version of film-critik.net was planned not only to bring changes in the layout and features improvements but also to prepare for the launch of other websites that will be built around it to focus on other specific types of content related to movies that will not be part of the main website but that will be linked to it. That way, this family of linked websites can form a rich "cloud" of specialized websites.

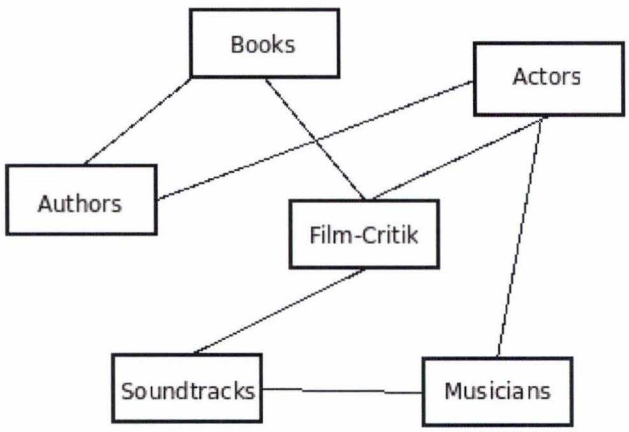


Figure 1.2: Cloud of specialized websites

These new websites will be very similar to film-critik.net. Although each of them will have some specificities that will make it unique and different from the others, many features will be shared among the sites.

Obviously this observation leads on the path of genericity and software reuse. While several similar products are foreseen, using a software product line approach allows to take advantage from the commonalities and focus only on developing what is actually different from what already exists on each new product. This approach brings an additional dimension to the code restructurations : a process which aims at anticipating and planning reuse of existing components.

The objectives of this thesis are

1. to explore software product line development techniques
2. to apply them to a real-life case study, namely Film-Critik.net,
3. to evaluate the results this application.

1.3 Structure of this document

This work is split in 4 parts. In part I, we first describe product lines and software product lines more precisely in order to give a better general idea and present the basic concepts of this paradigm in software development as well as the notations that will be used.

Then, in part II, methods and framworks used to build product lines are be reviewed and the method that will be applied on our project is described.

Part III presents the pratical application of the methodology on the case of the websites, then in part IV, the method that was used is reviewed for potential improvements and the conclusions of this work are presented.

Part I

Background

Chapter 2

Reuse in software development

In the first section, this chapter presents briefly how developers have reused existing code since they have begun to build software. Different methods were invented and developed, but unfortunately, they were not always used very efficiently. The second section presents various reasons why programmers have failed and still fail to reuse software properly.

2.1 History

From the earliest days of software engineering, reusing what already exists has always been a concern. Programmers first started by copying and pasting sections of code. With time passing by, some better methods have appeared.

- 1960's : subroutines, procedures and functions
- 1970's : modules
- 1980's : objects (Pol06)

  Early object oriented languages, SQL DBMS

  More portable than machine specific assembly languages. Limited reuse, math libraries, etc ...

- 1990's : components and code libraries (Pol06)

  Mature 3rd generation and OO languages, GUI and event driven programming, first portable compiled software packages, shared and dynamic link libraries (MFC, Microsoft Foundation Classes, Java). Components can be compiled separately and linked at runtime (COM, CORBA, etc...). Component based development processes.

- 2000's : services (Pol06)

  Web services, SOA, domain specific languages, reuse of functionality rather than code. Example : .net framework

Unfortunately, the archaïc and opportunistic "copy/paste reuse", which is also sometimes called "software cloning", "code scavenging" or "code salvation" (Bos00), is still one of the most widespread methods today.

The figure 2.1 shows how the level of reuse increases with the development paradigms developed over the years.

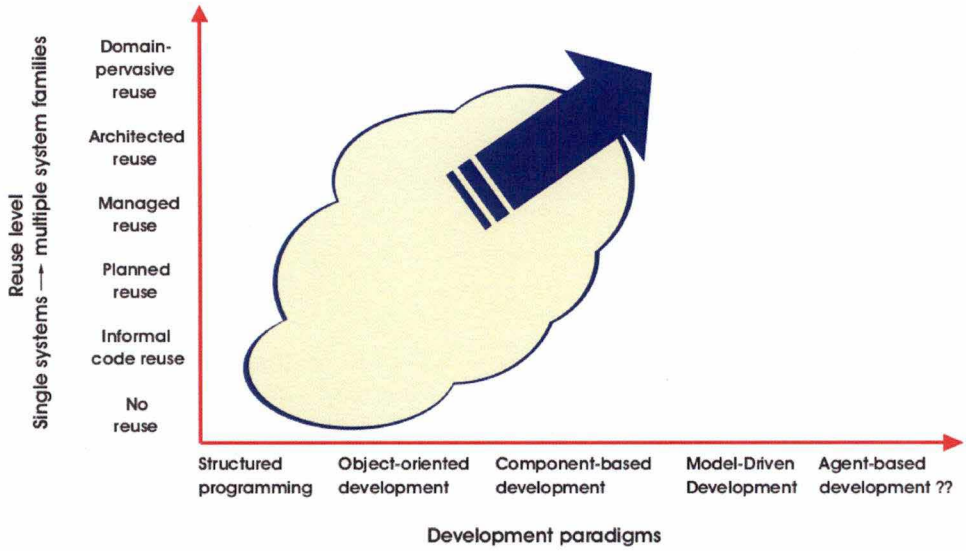


Figure 2.1: Development paradigms and reuse level (Ber07)

2.2 Why software reuse failed (and why it still fails today)

According to (KMB06), several forces lead to cloning :

- Developers have access to the source code.
- Cloning is expedient. It is easier to copy and modify than extending or generalizing the original source code.
- Cloning requires less coordination. Generalizing existing components requires coordination with the users of these components as well as the component supplier.

It is obvious that reusing code in this way will make maintenance more complicated. Fixing a bug that exists in the original source code requires to adapt every copy of this original source code.

However, any person who has actually tried to reuse existing components can tell you that there are countless reasons that will make them incompatible with the code that needs to call them. Existing components can use different units, dimensions, coordinate systems, formats, data validity assumptions, etc.
According to Jeffrey Poulin in (Pou96), developing a reusable component costs about 50% more than a one-off component. As the software projects are often budgeted and planned only to meet their own needs, allocated resources and time are often insufficient to build highly reusable components.
In (CN01), Clements and Northrop identify two cultural factors that play against a successful reuse of components :

- "Not invented here" : Software developers like to build their own stuff better than trying to reuse someone else's
- Risk aversion : If an adaptation needed to integrate the reusable component with the developed system could take longer than planned and impact the overall project planning, even a relatively small probability will suffice to reject the reuse option.

In (Sch99), Douglas C. Schmidt identifies other non-technical impediments to successful reuse, which include among others :

- Psychological impediments : developers perceive "top down" reuse effors as a lack of confidence in their technical abilities
- Administrative impediments : reusable assets are hard to catalog, archive and retrieve, especially in large organizations. Developers have a hard time locating reusable assets outside their workgroup.
- Economic impediments : supporting a reuse group has a financial cost that organizations find hard to fund.
- Political impediments : internal rivalries among units within an organization may prevent reuse of assets developed by other groups which can be perceived as threats to job security or corporate influence.
- Lack of skills : Developers might not have the technical skills and the knowledge of fundamental design patterns which makes it hard for them to create and reuse frameworks and components effectively.

2.3 Summary

Over the time, methods and paradigms have been developed in order to reuse existing pieces of software more effectively. Each new method and paradigm goes a level higher in terms of reuse and abstraction. However, despite all the efforts put in developing these technologies, several factors play against a systematic and planned reuse. In the next chapter, we'll see that industrial production of material goods do not suffer from these problems and go one step beyond everything that was described here in terms of reuse : product lines are thought from the beggining with reuse in mind. Software product lines apply the same principles to production of software.

Chapter 3

Product lines

This chapter first introduces the product lines and the underlying concepts that allow them to take advantage of the initial investments and efforts to build new things using what already exists in a planned manner.

Then, section 3.2 introduces software product lines and shows how the concepts of products lines have been translated to the world of software engineering.

3.1 Introduction to product lines

Fordism introduced the concept of mass production by exploiting the division of labor, by splitting the process of building automobiles in a number of very small, simple and repeatable processes performed sequentially along a production line. Each worker along the production line assembled the same pieces in the same way on every product that was presented in front of him.

The production process was well defined and structured. This allowed to guarantee that every product arriving at the end of the product line was built exactly in the same way as every other one.

Not only did this change in the production process improved productivity, but it also radically reduced the costs because every product that was built on the production line was taking profit of the investments made in the production process.

3.1.1 Variability and mass customization

Production lines and production processes have been improved further to allow a single production line to build similar yet different products using the same production process. That is, depending on the wanted product, a worker on the product line uses a particular piece among a set of interchangeable ones. The most obvious example on a car production line is probably the color of the car that is produced. The painting process is exactly the same whether the car to be produced is black or white. The worker simply perform his painting work using either the black paint when the car has to be black or the white when the car has to be white. In this case, only two different products can be built, but if choices have to be made at several places in the production process the number of different products that can be built can grow really fast. For example, if two steps in the production process allow 3 choices each (let's say 3 different colors and rim models are available), 9 different types of cars can be produced using the exact same production line and process.

3.1.2 What are product lines ?

A product line is the set of all similar yet different products that can be built using a given production line, its associated production process and the interchangeable pieces at each stage where a choice has to be made to define the final product.

3.1.3 Examples of product lines

- Automobiles, planes, etc.
- Engines
- Newspapers

3.2 Software product lines

Software product lines apply the concepts of fordism to software production. That is, using the same approach as "industrial" product lines, mass production and customization of software can be achieved through a well defined process.

3.2.1 Definition

"A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way." (NCB+09)

- Set of features

  Features bring a solution to fulfill the requirements. They are the means by which the goals can be reached.

- Common

  As products share similar needs, requirements and goals, they also share the features that fulfill them to reach the same goals.

- Managed

  The features are elicited, analyzed and managed to ensure that once a feature exists it can be reused and does not need to be developed again.

- Common set of core assets

  Core assets are the physical elements that implement and concretize features. As illustrated in 3.1, they are grouped in the "core asset base", a set of assets shared by all the products of the product line. Even if each product does not make use of all of the core assets, all of its assets come from the same core asset base which is unique to the product line.

- Prescribed way

  Each core asset has its own associated processes that prescribe precisely how it is used and assembled with other assets.

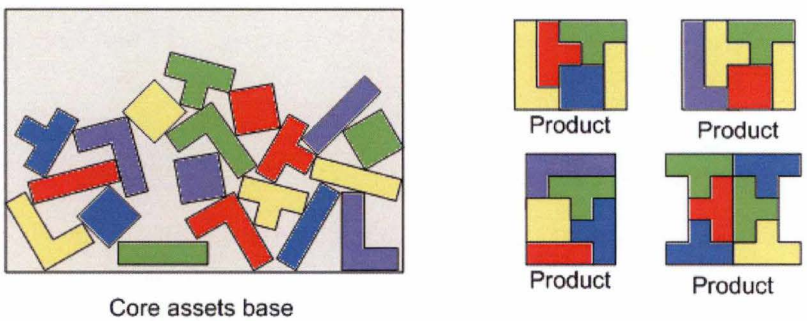


Figure 3.1: Core asset base and products

3.2.2 Motivations

Building a product line is not straightforward, it requires investments and changes in the organization, but it can really be worthwhile. The advantages brought by traditional product lines are transposed in the software world when building a software product line :

- Efficiency

  Even if they are not built nor launched together, knowing that several resembling products will have to be developed enables to perform some optimizations to achieve some scale productivity gains by :

  - regarding them as a family and conceiving the products as variations of a common architecture by designing and developing only once what is common and then focusing on what actually changes and what is specific to each product.
  - planning a repeatable and automated production process that will reduce the amount of time and errors between the development and the deployment of a product.

- Quality

  As the architecture and the components are reused they are much more used (and thus tested) than one-off software elements. This ensures that more bugs can be fixed and that the bugs that were fixed before are not repeated in new products. Moreover, once a bug is fixed, a new version of the products that were bugged can easily be released.

- Predictability

  Building a new product is a matter of following a well known and defined process. Once the product line is launched and that the first products have been released, new products can reuse existings assets and less time is spent on integration jobs that have already been done or automated before. This allows to predict more precisely the time needed to build a new product.

- Reduced time to market

  Planning reuse of existing assets from the beggining ensure that assets will actually be reusable across products and that everything that has already been thought about and developed won't be thought about and developed once more. Thus, most of the time needed to develop and launch a new product is spent in building new things, and no time is lost in developing the same things several times and fixing the same errors that have already been encountered before.

3.2.3 Efforts

The figure 3.2 compare the cumulative efforts and investments needed to build more and more products, between a product line approach and a traditional approach.

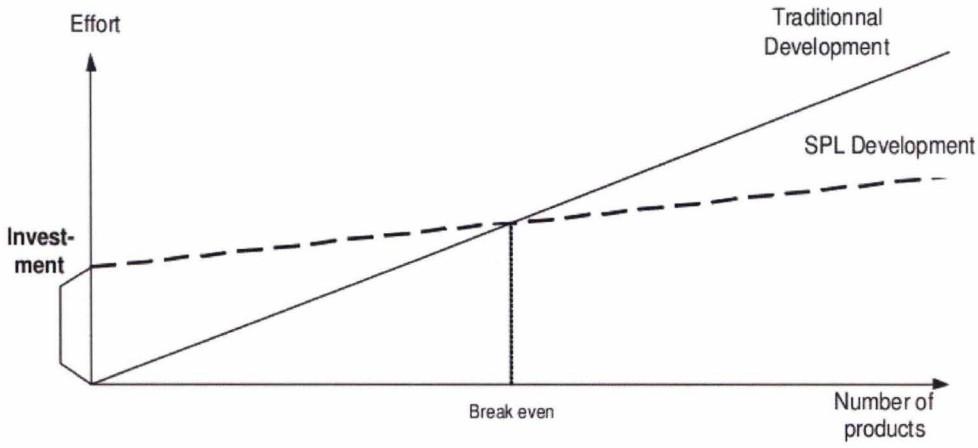


Figure 3.2: Cumulative investments needed to build new products (TH03)

At first, software product line may be a heavier solution, because a lot of things have to be thought about, analyzed and developed in a generic fashion.

However, it pays off in the long run : once the product line is running, less effort is needed to develop each new product, and past the break even point, the software product line approach is more interesting.

3.3 Summary

Product lines as well as software product lines capitalize on commonalities and take advantage of well defined processes and interchangeable pieces to improve efficiency, quality, and predictability and to reduce time to market. Of course these are only a few examples of the what can be achieved. On the other side, starting a product line requires more efforts and investments than developing a single software, but once it is running, new products are developed at lower cost and economies of scale make the product line more interesting. The higher initial efforts needed come from the fact that the variability, (i.e. the differences between the products) has to be taken into account from the beggining.

Chapter 4

Variability

As software product lines capitalize on commonalities to achieve large scale economies, variability is obviously an important part that has to be studied carefully to ensure that it will be possible to develop different products on a common basis. In this chapter, variability will be presented under different angles. We will first define what variability is in section 4.1, then in section 4.2, we will see why variability exist and why its importance evolves in software engineering. The next sections will focus on how and when variability is handled and implemented (sections 4.3 to 4.7).

4.1 Definition

According to Bachman and Clements in (CB05), variability is "the ability of a system, an asset, or a development environment to support the production of a set of artifacts that differ from each other in a preplanned fashion.".

In other words, analyzing variabilty in the context of a software product line involves determining what varies and how it varies between the products of a product line.

4.2 Reasons

As stated by Jan Bosch in (Bos), variability in software is increasing because of two reasons :

- Variability is moving from hardware to software
- Design decisions are delayed as long as possible

An obvious example of this is the software included in car board computers. Cars are built with the same mechanics and hardware and then configured, options are enabled or disabled to meet the client's request. Even the engine power is controlled by software.

4.3 Variation points and variants

A variation point is a particular point where a choice has to be made about a difference between products and how it is bound. The available options for a variation point are called *variants*, in other words, a variant is a realization of a variation for a given variation point.

As mentioned by Deelstra et al in (DSNB04), a variation point can be bound to zero or more variants and falls in one of these categories

- *optional* : 0 or 1 variant can be chosen

- *alternative* : a single variant must be chosen
- *variant* : 1 or more variants must be chosen
- *optional variant* : 0 or more variants can be chosen

Of course the choice of variants is always made out of a set of variants greater than the minimum number of variants that must be selected for if a single variant must be selected in a set of one variant, this becomes a mandatory feature and there is no real variability in this case.

In (DSNB04), the authors also define a fifth categorie for variation points : *value*. These variation points requires to choose a value within a given range. This range can be a restricted set of values as well as broader ranges like strings, integers, reals, or anything else.

4.4 Levels of variability

Variability can occur at different levels of abstraction, In (Bos04) Bosch identifies five levels ranging from the whole product family to the component implementation.

4.5 Sources of variability

In (BB01), Bachmann and Bass identify some sources of variation that lead to the creation of variation points :

- Variation in function : Some features may exist in some products and not in others, or with different characteristics.
- Variation in data : Data may be represented differently in different products (Different data structures, data types, etc.).
- Variation in technology : A product line may build different products to support different types of platforms, hardware types, OS, middlewares, languages, etc.
- Variation in quality goals : This type of variation occurs when products of different quality are build. High-end and low-end products do not use the same components. This can lead to variation in technology if the underlying technologies (frameworks, middleware, os, ...) are not suited well enough to support some quality goals.
- Variation in environment : Variation may occur when the way products communicate with their environment change. This is slightly different than variation in technology as this focuses more on the external systems with which the products will interact.

4.6 Binding times

The way of implementing a variation point depends much on when and how variants are to be added and bound to the variation point. The time when a variant is bound to a variation point is called *binding time*. A variant can be bound at many different times between development and runtime. Some binding times are

- *Development time*

  Variants can be bound at development time. For example, when developing a new component, and several implementations of another component are available. Deciding which one is used during development binds the given variant to the component.

Binding features during development is more "hard-coding", or implementing a design decision rather than truly making a choice about which variant to use.

- *Static code instanciation time*

  This is what is typically done with code preprocessor directives, like #ifdef in C. These directives tells the preprocessor to include some pieces of code and exclude others. At the time the preprocessor has done its job, the variant is bound.

- *Compilation time*

  The compilation time is very close to the static code instanciation time, as it happens just after. The difference is that the variant is bound during compilation. Static code instanciation time and compilation time are not always clearly bounded and can sometimes overlap, as the code instanciation and optimization can be considered as part of the compilation process.

- *Configuration time*

  Some variants can be bound during configuration time, this is sometimes the case for the variation points of type "value", which do not involve to select a component or a feature but rather configure a value.

- *Packaging time*

  Variants may be selected at packaging time, when including a particular file, library or some other physical asset in the package if a choice had to be made about which file, library or asset to include.

- *Install time*

  All available variants may be included in the package and the choice can be delayed to install time to decide which variant to install.

- *Startup time*

  The user is asked to make a choice before the program starts or when this choice is made based on environmental parameters. Startup time can be quite ambiguous as it might as well represent the first startup as well as every startup of the product.

- *Runtime*

  Variants are chosen during execution, on basis of any information that is available at the time the choice is made.

These are only examples of binding times. Depending on the language, platforms, technologies, etc., some of them might not be relevant at all, and other binding not mentioned here times may exist.

4.7 Variability realization techniques

As mentioned before, variation points can be implemented differently depending on the binding time chosen for a given variation point.

In (SvGB05), the authors analyze techniques that can be used to realize variabiily at different binding times :

- Condition on constant

  This technique consists in defining a constant, and including or excluding parts of the code by surrounding them with conditional instructions based on this constant. This can be done using code preprocessor directives (in which case the binding time can be static code instantiation time) or using "actual" constants in the code itself (in this case, the binding time can be compilation time or static code instantiation time if the compiler performs an opitimization phase to remove unused sections of code)

- Code fragment superimposition
  This involves merging pieces of code together, either by instanciating a template of code or by using other techniques like the ones used in aspect oriented programming (source code weaving, binary patching). Again here, the binding times can either be static code instanciation time or compilation time (or even "post-compilation" time) depending on how the code is generated.

- Condition on variable
  This techniques resembles very much the condition on constant except that the conditional instructions are based on a variable which can change at various times. In this case, the variable might be set once at startup time (on basis of configuration or environmental parameters) or can vary during runtime.

- Binary replacement
  Replacing a binary can be done in several ways :

  - By using linker directives : in this case the variant will be bound at compilation time (or "linking time") and won't ever change after.
  - By replacing physical binaries : in this case, the variant can be bound at different times depending on how the product works, it can be loaded once at startup time or can be replaced during runtime. This technique can also be used at packaging time to include a specific binary in the package or at install time when a specific binary is selected to be extracted from the package (or downloaded from the Internet, or provided by the user for example) to be installed.

These four techniques cover all the binding times described previously, but a lot of other techniques exist. Although it is an important parameter, the choice of the variability realization technique is not driven only by the desired binding time.

In(SvGB05), Svahnberg et al. describe also the following characteristics which must be taken into account :

- The introduction time, the moment the technique is introduced in the process
- The times when it is left open for adding new variants
- If the collection of variant is explicit (an exhaustive list exists) or implicit
- If the binding is done internally (by the system itself) or externally (by tools like configurators, compilers, etc. or "manually")

4.8 Summary

Variability is one of the corner stones of software product lines. It is what allows product lines to build several similar yet different products on a common basis. Variability exists and becomes more and more important because it is easier to implement different behaviours and features in software on the same hardware rather than changing hardware for each product. When analyzing products before starting a product line, variations between products are identified and variation points are defined for each variation. Depending on the choices that have to be made, variation points can be of different types (optional, alternative, variant, optional variant and value). Variability can be seen at various levels between the product line itself and the implementation. The time at which the decision of which feature or behaviour is bound to a given variation point is called "binding time" and can be anywhere between development and runtime. Depending on this binding time and other paramaters, different techniques can be used to implement variation points. The next chapter will explain how these variations are represented when creating features models that serves as basis when analyzing variability.

Chapter 5

Modelling features

Commonalities and variation points are modelled using feature diagrams. This chapter briefly introduces the notation used in feature diagrams presented further.

Features diagrams are trees where the nodes represent the features and the links represent the relationships between them.

5.1 Motivation

"The purpose of feature analysis is to capture in a model, the end-user's (and customer's) understanding of the general capabilities of applications in a domain" (KCH+90)

According to (KCH+90), feature models might describe different things like :

- services provided
- performance levels
- hardware platforms supported (or required)
- costs
- others

Feature models allow to formally represent variability, in other words, they allow to determine what is common and what varies between the members of a product line.

5.2 Decompositions

A feature is decomposed in different ways. Each type of decomposition can be expressed either graphically or using cardinalities.

5.2.1 Mandatory features

Mandatory features, also called sub-features, are required by the parent feature, which would be incomplete without them. They are represented by a simple link between the parent feature and the sub-feature, sometimes with a filled circle above the sub-feature.

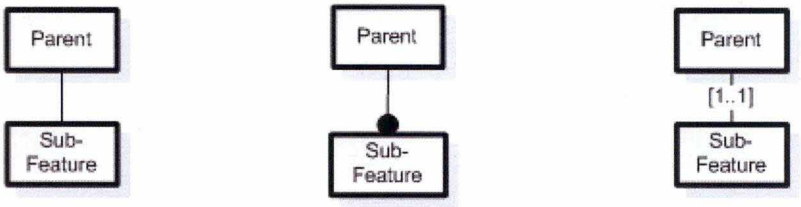


Figure 5.1: Mandatory sub features

5.2.2 Optional features

Optional features are not required by the parent feature, they simply can bring something more to it. They are represented by a link with an empty circle above the optional sub-feature. This is how "optional" variation points are represented.

Figure 5.2: Optional features

5.3 Compositions

In order to form products, features can be assembled according to certain constraints. Features can be taken together ("and"), or choosen using alternatives ("or" and "xor"). This is how "variant" variation points are represented.

- And

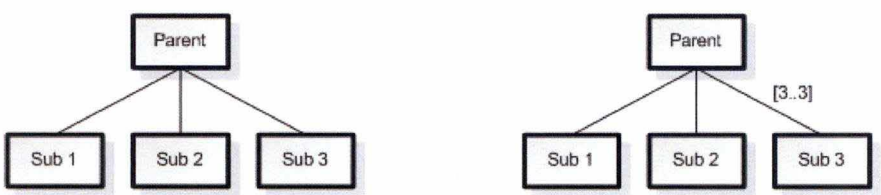


Figure 5.3: "And" composition

- Or

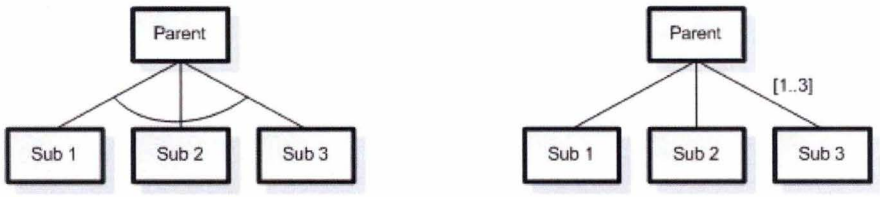


Figure 5.4: Alternatives ("Or" composition)

- Xor

Figure 5.5: Alternatives ("Xor" composition)

5.4 Constraints

Besides compositions, inclusion ("requires") and exclusion ("excludes") constraints can also be specified.

- Requires

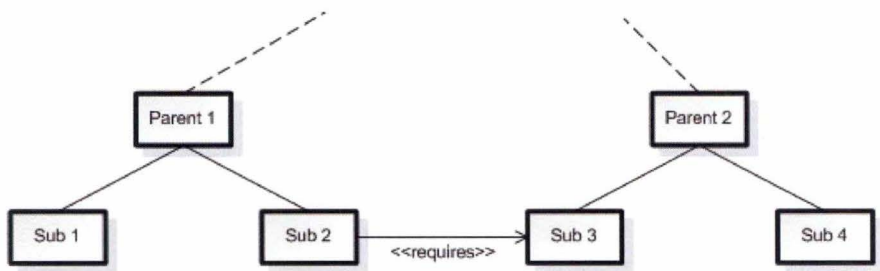


Figure 5.6: Inclusion

- Excludes

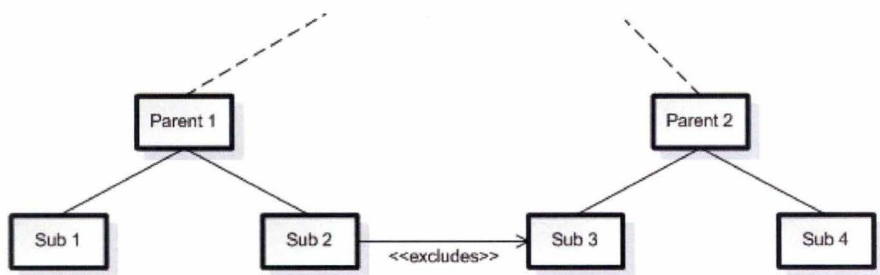


Figure 5.7: Exclusion

5.5 Values

Variation point of type "value" are represented using attributes. Only leaves of the tree can be of type "value" and use attributes.

Attributes are typed, either with a primitive type (integer, string, ...) or with a custom type described textually.

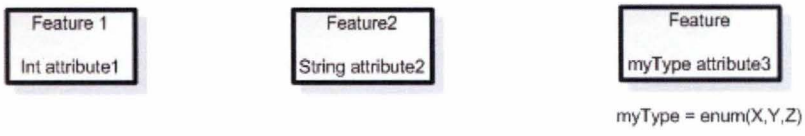


Figure 5.8: Attributes

5.6 Summary

In this chapter, feature models were summarily presented, then the notation used to design formal feature diagrams was presented, with the different possibilities for the different types of constructions that can be found in such diagrams.

Chapter 6

Modelling goals

This chapter introduces summarily i*, a language used to model goals and the relationships that bind them. Many concepts and elements that are not essential to the understanding of the diagrams presented in this document are voluntarily omitted.

This whole chapter is based on knowledge available in (AHYG07).

6.1 Motivations

Most common requirements engineering techniques nearly always deal with "what" the system being designed must do and "how" it must be done. However, few methodologies actually include an analysis of the goals, of "why" the system is built. I* (or i-Star) is a language that provides a way to create a formal goal model.

6.1.1 Definition

"A goal is an objective the system under consideration should achieve. Goal formulations thus refer to indented properties to be ensured; they are optative statements as opposed to indicative ones and bounded by the subject matter" (vL01).

6.1.2 Reasons to model goals

In (vL01), van Lamsweerde presents various reasons to model goals :

- To achieve requirements completeness and to determine if the specified requirements are sufficient to achieve all the goals
- To avoid irrelevant requirements. If a requirement does not meet any goal, it is probably useless.
- To provide a "traceability link from high level strategic objectives to low-level technical requirements".
- To detect and resolve conflicts among requirements
- To drive the identification of requirements

6.2 Nodes

6.2.1 Actors

Actors are "intentional entities", in other words, they have intentions, they want to reach goals. An actor's boundary contains the goals an actor wants to reach, the actions he performs and the resources he uses.

Figure 6.1: Actor (AHYG07)

6.2.2 Elements

Elements can be of one out of four types :

Goal

A goal (or "hard" goal) is something an actor wants to be realized. It is defined sharply and is either satisfied or denied.

Figure 6.2: Hard goal (AHYG07)

Soft goal

A soft goal is similar to a goal, except that there is no sharply defined criteria for this kind of goal. It can be partially satisfied. The level of satisfaction (sufficient or not) is left to the actor's appreciation.

Figure 6.3: Soft goal (AHYG07)

Task

A task is an action (or a group of actions) an actor desires to carry out in order to reach one or more goals.

Figure 6.4: Task (AHYG07)

Resource

A resource is something (physical or not) an actor wants or needs in order to reach one or more goals.

Figure 6.5: Resource (AHYG07)

6.3 Links

6.3.1 Dependencies

A dependency exist between two actors or elements when the actor at the source of the link depends on the actor at the end of the link to satisfy a goal, accomplish a task or obtain some resource.

6.3.2 Means-End

Means-End links a means to the end it helps to reach.

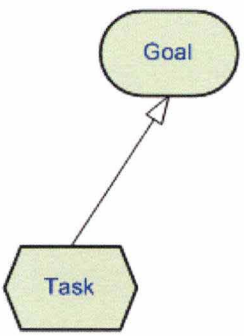


Figure 6.6: Means-Ends link (AHYG07)

6.3.3 Decompositions

A task can be decomposed in various ways, in each case, the element that is at the source of the link is the task being decomposed, and the element at the other end is the part. In most cases, the two elements will be tasks (meaning that the completion of a higher level task needs some lower level task to be accomplished), but goals, softgoals and resources can also be part of a task.

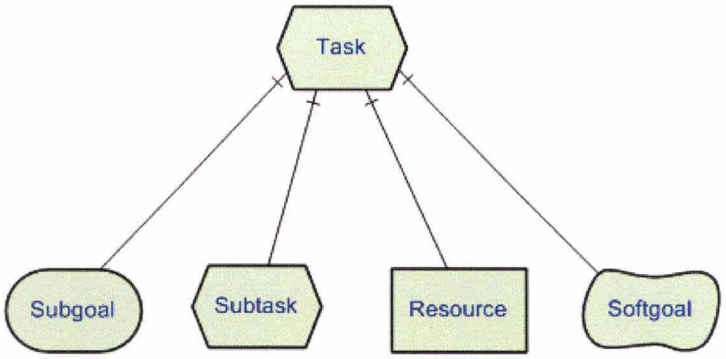


Figure 6.7: Task decomposition (AHYG07)

6.3.4 Contributions

Each element can contribute to satisfy a goal or a softgoal. Contributions can be positive or negative with different levels, or unknown.

Positive	Negative	Unknown
Make	Break	Unknown
Some +	Some -	-
Help	Hurt	-

Table 6.1: Contributions

Contribution levels In (AHYG07), the authors define the contribution levels as follows :

- Make and break contributions either satisfy or deny fully a goal.
- Some + and Some - contribute partially and may be sufficient to satisfy or to deny a goal
- Help and Hurt contribute partially but are not sufficient by themselves to satisfy or deny a goal

And / Or compositions Contributions can also be composed using "And" and "Or" contribution links.

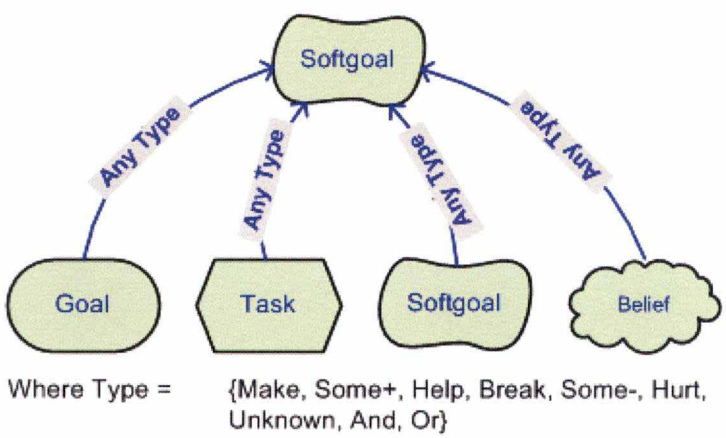


Figure 6.8: Contribution links (AHYG07)

6.4 Summary

In this chapter we have seen why goals are an important part of the requirements engineering and introduced the i* language to create goal models. The different constructions allowed in i* were briefly presented.

Part II

Methodology

Chapter 7

Methodology

In this chapter the framework from the Software Engineering Institute (SEI) of the Carnegie Mellon University will be presented summarily. We'll then analyze the opportunity to use it in the context of this work and introduces a high level view of the methodology and the development process of the product line presented in this thesis.

The framework from the Software Engineering Institute (SEI) of the Carnegie Mellon University describes many aspects related to the development of software product lines.

7.1 Software product lines development methods

7.1.1 FAST

FAST stands for "Family-oriented Abstraction, Specification and Translation". (Har02)

According to (Har02), this method is subdivided in 3 processes :

- Domain qualification
- Domain engineering
- Application engineering

These processes follow each other and are subdivided in activities which produce artifacts. The application engineering process can provide feedback to the two other processes, which can in turn impact the application engineering back. (Har02)

In (TH03) and (Ber07), the authors mention that this method focuses on providing a systematic approach to potential product families and that it describes the need for abstractions that remain stable during the entire lifecycle of the product line. They also mention that the domain analysis and the specification of requirements is too complex because they need to be extremely precise in order to generate code.

7.1.2 FODA and FORM

FODA stands for Feature Oriented Domain Analysis (KCH+90)

The FODA method suggest three phases : (KCH+90)

- Context analysis
- Domain modelling
- Architecture modelling

FODA introduces the features diagrams and may be useful for modelling features but it does not provide an adequate method to build software product lines.

FORM (Feature Oriented Reuse Method) is an extension of FODA which adds processes that focus on the development of core assets and the definition of a a parametrized reference archicture and reusable components (TH03).

FORM brings interesting additions to FODA, which could prove to be useful in the context of this project.

7.1.3 RSEB

RSEB (Reuse-driven System Engineering Business) makes use of the use cases of the Unified Modelling Language (UML) (TH03).

It derives from the traditional split between domain and application engineering and propose the following separate activities (VK00):

- Application Family Engineering
- Component System Engineering
- Application System Engineering

In (VK00), the authors mention that "Application Family Engineering" and "Component System Engineering" can be considered as variations of the domain engineering.

7.1.4 FeatuRSEB

In (TH03), the authors describe FeatuRSEB as a combination of FODA and RSEB which adds feature modelling to the RSEB method. They also mention that this method merges the features diagram of FODA with UML diagrams by linking them together using UML dependencies.

7.1.5 PuLSE

The PuLSE method (Product Line Software Engineering) was designed by the Fraunhofer-Institut für Experimentelles Software Engineering (IESE) and is articulated around three main elements (BFK+99):

- Deployment phases
- Technical components
- Support components

In (BFK+99), the authors present PuLSE as a "customizable methodology for the conception and deployment of software product lines", which could make it an interesting choice to build our product line as we could customize it to meet the needs of the project. This is confirmed by Berger in (Ber07) where he mentions that "PuLSE is also suitable for small and medium-sized projects". On the other side, he also mentions that the "commercial character" and the lack of fully free documentation make this method unsuitable for his project which is similar to the product line we are building in many points.

7.1.6 KobrA

KobrA stands for "Komponentbasierte Anwendungsentwicklung" and is a "ready to use" and "massively tailored" customization of the PuLSE method (TH03) and (Ber07).

It splits the conception of the product line in the two traditional parts which are here named "Framework engineering" and "Application engineering" with their sub steps, and also defines processes for the implementation,

release and testing aspects (Mat04). In (Mat04), Matinlassi also mentions that "KobrA states it is a simple, systematic, scalable and practical method". This makes it suitable to a wide spread of project types and sizes which can include the product line developed from Film-Critik.

7.1.7 The "SEI Framework"

The "SEI Framework" (i.e "Framework for Software Product Line Practice") has been developed by the Software Engineering Institute (SEI) from the Carnegie Mellon University. It is an evolving document that is available on the Internet (NCB+09) and is completed by the book of Clements and Northrop (CN01).

It splits the development of software product lines in 3 essential activities and 29 practice areas.

Essential activities

In (CN01), the authors identify 3 essentials activities performed simultaneously as shown on figure 7.1 :

- Core asset development :
  The core assets set includes several very different types of assets. For example, a core asset can be, a component, the underlying architecture, a production plan, a test plan or lots of other things. The development of these assets is an important part of the product line development.
- Product development
  Products have to be defined precisely before they can be produced by the product line. Developing products on basis of existing core assets is an activity of its own that deserves a special attention.
- Management
  All the processes have to be managed carefully to ensure everything runs smoothly and problems are avoided.

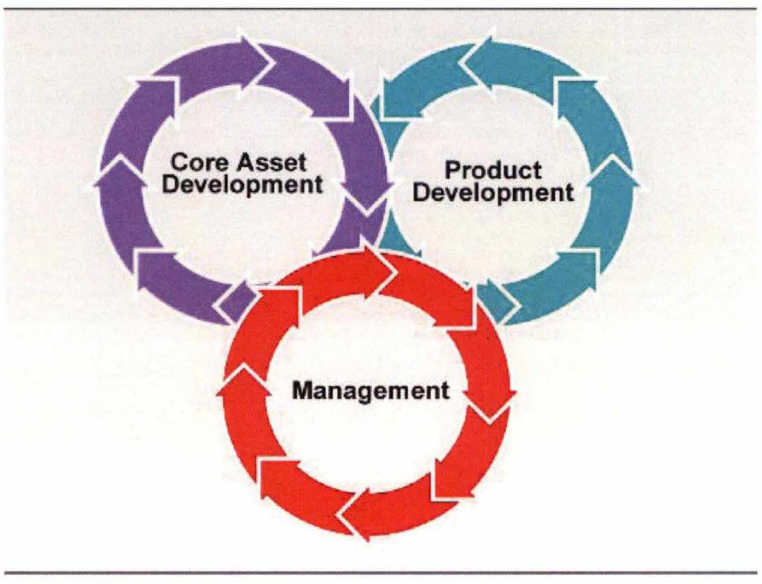


Figure 7.1: Essential activities (CN01)

Practice areas

"A practice area is a body of work or a collection of activities that an organization must master to successfully carry out the essential work of a product line."(CN01)

The framework also defines 29 pratices areas grouped in 3 categories :

- Software engineering :
  The practice areas found in this category focus mainly on the creation of software assets. Some of these practice areas are related to the "core asset development" essential activity, while some others are related to the product development activity
- Technical management :
  This category comprises the practice area that touches the 3 essential activities together.
- Organizational Management :
  These practice areas are related to activities described by the "management" part of the 3 essential activities.

Patterns

Clements and Northrop (CN01)also defined numerous patterns that describe generic solutions to common problems that arise during the development of software product lines. These patterns are in fact ways to apply and combine the different practice areas together in order to solve or avoid potential problems.

7.2 Selection of the method

The SEI framework is probably one of the most complete and best documented of all the reviewed methods and frameworks. It covers many essential aspects of product lines development. It is based on real life enterprise projects and other methodologies (For example, in (CN01), the authors references methods like FORM, FODA, PuLSE and other, to reuse their contributions). Each practice area is described extensively, with the related aspects peculiar to product lines and the associated risks, along with reference to further information if needed. It "provides a standard way of working and brings many solutions to particular problems independently of the development method that is used" (Ber07).

However applying completely all of these practice areas would require a team of several experimented people who can take care of several practice areas at once and communicate effectively as almost everything is interrelated. A pilot project which is intended to start a product line in a company might not receive enough support and resources (time, highly skilled people, ...) if the management is not commited enough to launch of the product line or if it can not afford to allocate the needed resources to the project. Besides that, for smaller projects like our case study Film-Critik, many of these practices area are not always useful or relevant (we don't need funding nor acquiring components for example).

In fact, when using this framework, one must remember that it is a framework and that, as Berger states it in (Ber07), "it describes the activities at an abstract level, which can be adapted to different development processes. It is not directly applicable that's why a tailoring to the environment of project to which it is applied must take place". This means that some projects won't need certain practice areas at all, and that a selection has to be made on basis of the needs of each project, small or big. It also means that the activities described in the framework are not a methodology that can be used "as-is" but that they can be adapted to almost every software product line in order to develop a custom practical methodology that suits best the needs of the project it is applied to.

The combination of this flexibility and adaptability to specific products and its completeness that offers a great choice of tools to guide the building of a product line are the two main arguments for using this framework as a

basis to create our custom method. We will select a few parts of everything that is described, and combine them together to form our own methodology.

7.3 Development methodology

As mentioned before, the "SEI Framework" describe a lot of patterns, that helps to solve specific problems and situations. One of them is the "adoption factory" pattern which describes the steps to take when adopting a software product line for the first time.

7.3.1 The "adoption factory" pattern

The "adoption factory pattern" defined by Clements and Northtrop in (CN01) define three temporal phases as shown on figure 7.2:

- Establish context
- Establish production capability
- Operate product line

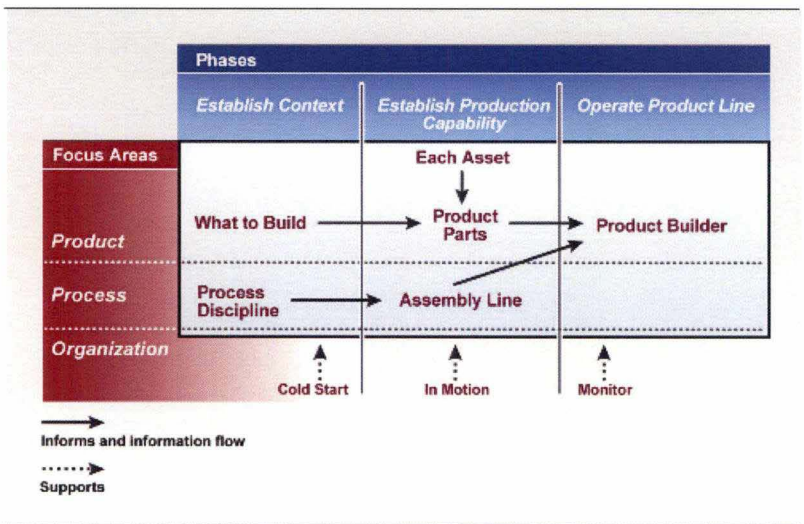


Figure 7.2: The adoption factory pattern and the temporal phases(NCB+09)

This pattern defines a combination of other patterns areas from which we will inspire the structure of this work without applying these exactly as described as they involve many practice areas that are not relevant to our case.

7.3.2 Development process

We will follow the two first phases :

1. Establish context :
   During this phase, we'll analyze and present the context in which the product line is going to be built, why it is built and the external constraints that apply to the development. This phase is described in more details in chapter 8.

2. Establish production capability :
   This phase will focus on the development of the building blocks and on the processes followed to assemble these blocks together to form end-products. This phase is described in more details in chapter 9.

7.4 Summary

In the first section of this chapter, we have first presented and reviewed briefly different methods that can be used to build software product lines. We have seen that even if each of these methods and frameworks brings its own original contribution, most of them are based on the same principles and concepts (which are named differently depending on the method) : they first focus on analyzing the context and the domain, and then on the applications that are going to be built. Some of them give a special importance to the software engineering aspects while others balance also focus on the management aspects.

The framework from the SEI of the Carnegie Mellon University has been introduced more in detail and its choice as reference method to guide the rest of this work has been discussed in section 7.2. The section 7.3 describes the methodology that will be followed and the development process which is based on the temporal phases of a pattern and concepts from the SEI framework.

The next chapters describe more in detail the two phases of the development, which are in fact the same as the two traditional parts of the methods reviewed in this chapter.

Chapter 8

Establish context

Establishing the context is the first step to take when developing a product line. It allows to know more precisely what we are going to build and which constraints exist. This chapter introduces the different elements of the context that must be detailed and how to describe them.

8.1 Domain model

The domain model describes the concepts and shows how they are related to each other. It is often expressed as a conceptual diagram with textual comments or annotations.

8.2 Goals of the product line

Describing the goals of the product line, the reasons why the product line is built helps to discover and define requirements. This is can be done as a textual description, possibly with a diagram if necessary.

8.3 Scope and requirements

The scope of the product line defines which products are "in" and which are "out" of the product line. In other words, it describes the products that will constitute the product line or that the product line will be capable of including. (CN01)

These products are typically described as a set of commonalities which exist among the different products, and the ways each product varies from the others. The scope might also describe constraints and dependencies existing between software components, features, etc. These constraints may be purely technical (for example, one component depends on an other to perform its work) or may be imposed by the business itself (legal constraints are an example of this)

The scope of the product line has to be defined very carefully for if it is too large, assets won't accomodate the variability, complexity will increase, and the development will be exactly like it is for a traditional one shot product development. On the other side, if the scope is too small, the product line might not be generic enough to accomodate the future growth and no economies of scope will be possible. (CN01)

8.4 Organizational context

The organizational context describes organizational dimensions and constraints that apply to the development of the product line. This may contain obligations to handle backwards compatibility with legacy systems or older

versions or describe how products will be made part of the product line (all at once, one at a time...).

8.5 Technology forecasting

Technology forecasting is an activity where the technologies (languages, frameworks, libraries,...) that are utilized and that will be (or may be) utilized in the future are studied in order to ensure compatibility between the product line and the future technologies.

8.6 Summary

In this chapter we have seen what important elements must be detailed in order to describe the context of a product line. First of all, the domain model describes in which field we are working, then answers to the why and what questions are answered through the goals and the scope. Organizational context and technology forecasting mention some additional constraints that will influence the whole development process at various levels.

Once we know what we are going to build, the next step is to establish the production capability, that is, to give ourselves the means to develop the products effectively. That is what the next chapter is about.

Chapter 9

Establish production capability

The production capability includes all of the elements that are needed to actually build a product. It includes several core assets, particularly the architecture and the components that are assembled to form products, a production plan to define how everything is assembled and a production line that can take various forms.

Establishing the production capability is the main goal of the "core assets development" activity (NCB+09). This chapter describes these assets and how they are going to be developed.

9.1 Architecture

The architecture is the common base around which every product is built. It contains everything that each and every product needs, no less, no more. Moreover, the architecture must be designed very carefully for it is the central artifact upon which everything relies. A badly designed architecture can prevent every product built upon it to meet certain quality requirements (like performance or security for example).

The architecture exposes the interfaces and the variation points on which reusable components will be plugged. These are the points where variability will be allowed and constrained. The choices made here have a direct impact on the extensibility and complexity of the system, and thus, on the possibility to reuse the architecture to build new products. Constraining variability and extensibility allow to lower the complexity but may prevent or complicate future reuses of the architecture, while allowing too much variability may increase complexity and make development of reusable components harder.

9.1.1 Choice of the variability realization technique

How variability will be implemented is not a straightforward decision. In (SvGB05), the authors mention that several parameters are involved, like :

- When are new variants added to the system ?
- When is the system bound to a particular variant of a feature ?
- Does the system know what variants are available ?
- Is the choice of the variant made by the system itself or by an external entity ?

9.2 Reusable components

Everything that is part of one or more products and that is not included in the architecture is implemented using components that can be plugged onto the architecture to enhance it with new features, to add new behaviours.

Usually reusable components implement

- Optional features : it is easy to include or exclude the feature by including or excluding the whole component that implements it.
- Varying features : the right variant can be selected among others to fit the needs of the product

Of course, reusable components can be used as parts of an architecture.

To be reusable, components should :

- Be generic enough to support the variations across products
- Offer a stable interface to communicate with the other components that might use them.
- Minimize coupling with other components (loose-coupling).

9.3 Products description

The product description is created by making a lot of decisions :

- For each variation point defined by the architecture, a variant has to be selected.
- Each variant is bound to a specific set of assets that may need to be configured.

This is the product development. The product description contains all the choices made during this process.

9.4 Production plan

The production plan is "The guide to how products in the software product line will be constructed from the product line's core assets." (CN01)

In (CM02), the authors mention that the production plan should define (among other things):

- The product development process
- The inputs needed to build a product
- The activities that result in a completed product and how they will be performed :
  - Identifying specific assets required to build the product
  - Developing new components if needed
  - Performing customization
  - Integrating components

As core assets have attached processes which describes how they are reused and integrated, the production plan merely assembles all this processes together in the right order as illustrated in figure 9.1.

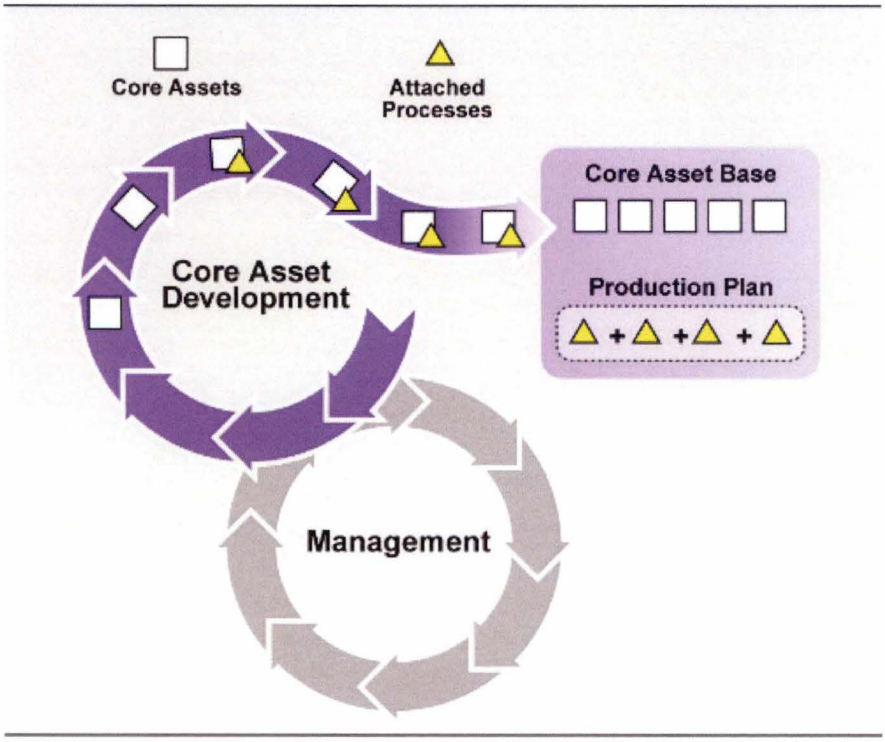


Figure 9.1: Production plan assembles attached processes (NCB+09)

9.5 Production line and tools

The production line is made of tools which help to build the products by automating repetitive parts of the production plan, or possibly the whole production process.

Depending on several factors, like technologies used in the project, availability of tools, compatibility of the available tools with the core assets, etc. choices will have to be made about the tools that will compose the "production line".

9.5.1 Utility

Tools of the product line can be used to automate or simplify various types of operations :

- Configuration
  A configuration tool helps to create a valid configuration to build a product. It helps in the choice of variants by presenting available choices and verifying that constraints are respected. It can generate a configuration file in a given format that can be used by a generation tool.

- Generation, integration, packaging
  Once a configuration has been defined, a generation tool can assemble assets on basis of configuration information to build a complete product. Assets can also be generated on basis of templates and/or configured using information found in the configuration. Of course, such a tool will need more information than the set of selected features for a particular product, it also needs to know how features and assets are related, in other words, which assets are needed for a given feature.

- Deployment
  If needed, and if the configuration file contains the required parameters, a tool can handle the deployment of the product where it needs to be done. This can be especially handful if the deployment is long and complex,

if it contains many steps or different systems, etc, ... A tool will accelerate the process and prevent to make mistakes and to forget steps.

9.5.2 Choice of the tools

Depending on the particular needs of the project, and on the parts of the production being automated, tools will have to be choosen and / or developed. Generic tools are easy to find, but are not necessarily well suited to the project, nor compatible with each other. Tools can also be customized if possible, adapters can be built to make tools compatible with each other when they aren't or tools can even be developed specifically to fit perfectly.

9.6 Summary

In this chapter, the assets needed to build products from the product line have been detailed. First of all, the architecture and its variation points, then the reusable components that will be "plugged" on the architecture to form products, this is also where the connection is made between features and assets. Based on the description of the products in terms of features, we now know what assets are needed to build a given product. The production plan describes the processes by which all these assets are assembled together. These processes are highly repetitive and can be automated using various tools that will speed up the process and avoid making mistakes.

The next chapters put all these steps in pratice on our case study, the software product line built from the film-critik website.

Part III

The product line

Chapter 10

Establish context

This chapter describes the first steps towards the development of the software product line that we intend to build in this work.

We will first describe the domain and the goals (sections 10.1and10.2), then in section 10.3we will describe the different features and determine which are common to all products and which vary. We will also see how features contribute to the achievement of the different goals identified (section 10.4). Finally, sections 10.5 and 10.6 will describe some organizational and technological constraints.

10.1 Domain model

The product line will contain products that are all related to a common domain : the cinema and everything that revolves around it, that is, movies of course, actors, directors, soundtracks, musicians, characters, ...

The first element that started the creation of this model is the "Movie", because it is the main subject of Film-Critik.net, the first website that serves as basis and first product of the product line being built. A movie page about a movie also contains other information, like the country(-ies) the movie is from, the actors that played a role in the movie, the people that directed it and the company that distributes it.

Next to these basic information, it is now wanted to be able to link to other information that might be relevant, like a list of the characters, the original soundtrack of the movie, and the book that inspired the movie if it exists.

These new type of information can also be presented as main subject and reviewed. As the schema shows it, information about a book contain the characters which appear in the book and the author who wrote it. A soundtrack is played by a musician or a music band which groups musicians.

Authors, musicians, their band, as well as actors and directors are all artists and have an origin country.

10.2 Goals

This section describes the goals of the product line. In order to keep schemas as readable and concise as possible, these goals have been grouped in several distinct categories.

The first two categories describe the final goals of the product line owner and the website owner :

- Product line approach
- Internal goals

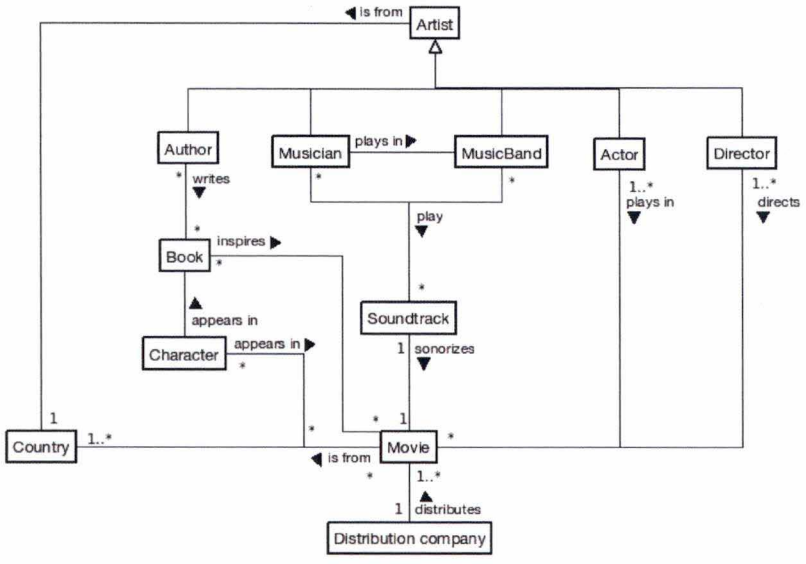


Figure 10.1: Domain model

The last three categories describe the means to reach the goals, grouped by the other actors involved in relationships with the website owner :

- Visitors
- Partner websites
- Distribution companies

10.2.1 Product line approach

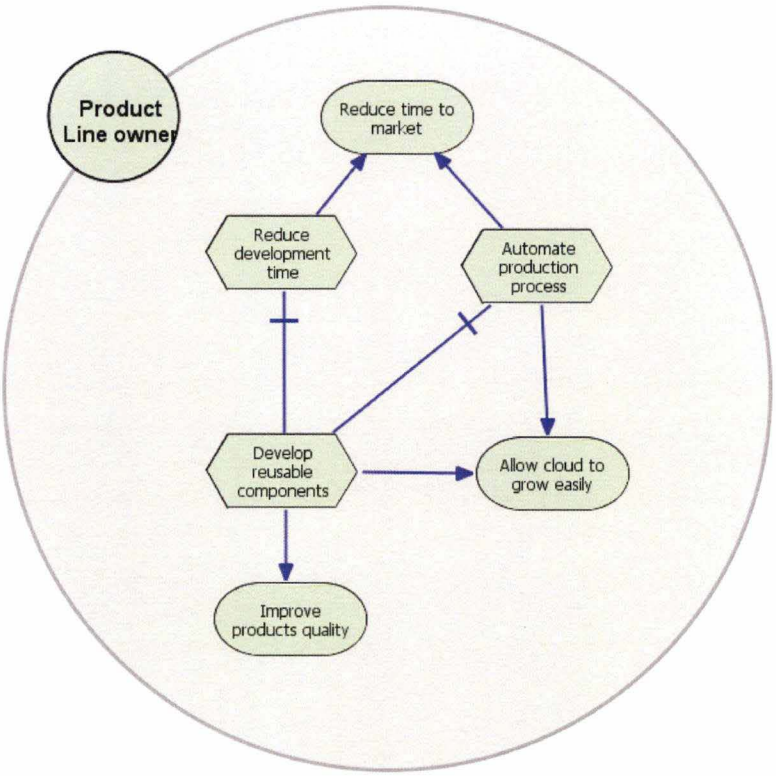


Figure 10.2: Product line approach goals

- Develop a core software asset base that will allow software assets to be reused easily (*"Develop reusable components"*) in several products instead of being rewritten for each new product (*"Reduce development time"*).
- Improve products quality and, as a consequence, the satisfaction of the visitors. In the long run, this will also limit maintenance time and the number of bugs to fix in each new product.
- Reduce time to market, when a new product is desired, the time to build it should be as short as possible.
- Enable the cloud of websites to grow easily, that is, simplify the process to create new websites and integrate them together.

10.2.2 Internal goals

This describes the "internal" goals of the websites, i.e. higher level goals and the ways to satisfy them.

The main goal is to get paid for advertising. This happens by selling ad space and get ads clicked for "pay-by-click" advertising systems.

Popularity and traffic

The popularity of the website, and especially the number of visitors (*"Grow traffic"*) is one of the most important (and hard to get) assets needed to make money from a website. The more visitors a website has, the more the ads it displays are seen, the more it is likely to drive traffic to the websites it links to, the better these advertising spaces and links can be monetized.

Traffic mainly comes from :

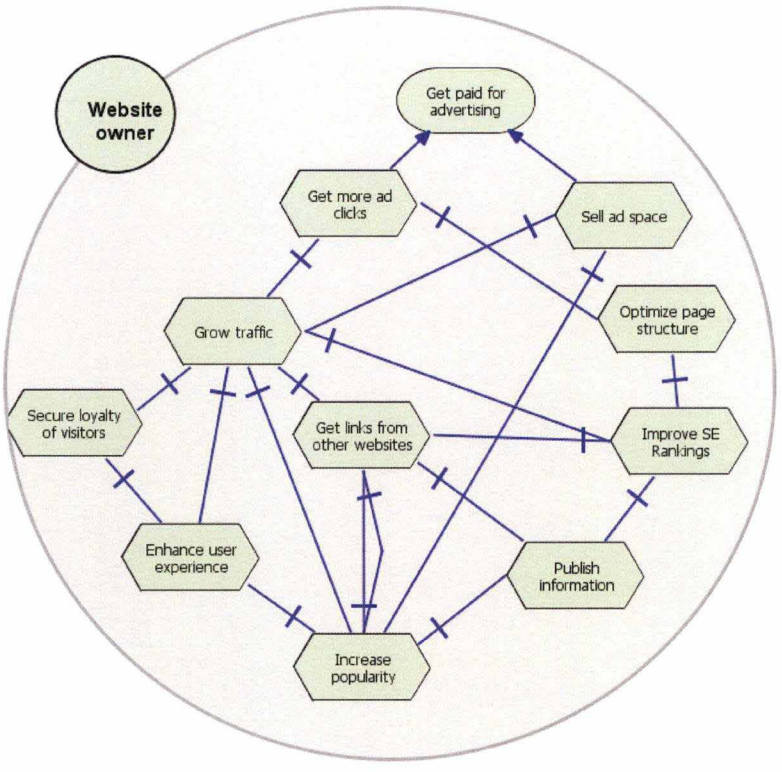


Figure 10.3: Internal goals

- Search engines ("*Improve SE Rankings*")
- Other websites ("*Get links from other websites*")
- Loyal / recurrent visitors ("*Secure loyalty of visitors*")

Another reason to want many visitors, is that a very popular website has a big potential influence. Being able to publish an information that will be read by thousands or millions of people gives a kind of limited although not null power some webmasters might want.

Page structure

The structure of the web pages plays an important role in search engines ranking, ("Improve SE Rankings") that's why optimizing it is important ("Optimize page structure"). A page that is well ranked for common queries in search engines is more likely to be visited than a page that appears in the last results.

The structure of the pages, and the positioning of the ads has an influence on the click rate ("*Get more ad clicks*"), again optimizing the page structure is useful here.

10.2.3 Visitors

The following schema describes goals which involve dependencies from or to a visitor. A visitor is any person which visits one of the websites of the product line, occasionally or on a regular basis.

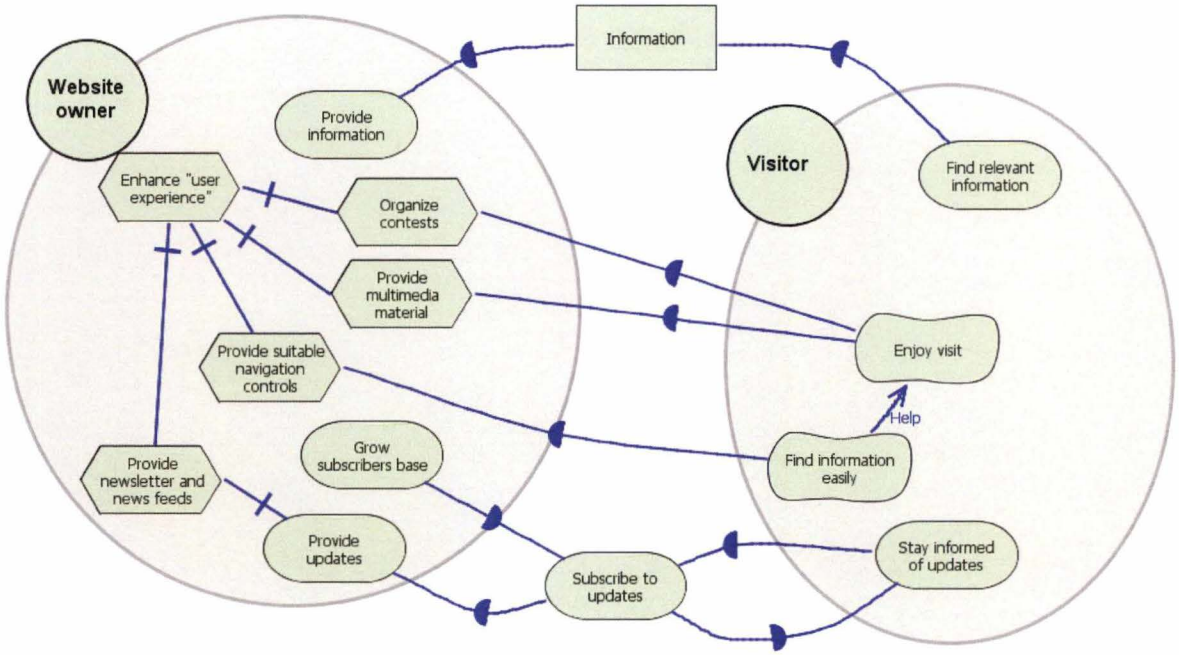


Figure 10.4: Goals related to visitors

Information and navigation

A visitor visits a website above everything, because he's looking for information about the subject of the website. His main goal is thus to "*find relevant information*". In order to allow him to satisfy his goal by visiting the website, the website owner has to provide him the information he's looking for ("*provide information*").

However, the information might well be available on the website, but if he does not find it quickly enough ("*Find information easily*"), the website will be of no use to him and he'll leave it rapidly, so the website owner has to provide him with the means to reach wanted information easily ("*Provide suitable navigation controls*").

User experience

As the website owner wants his visitors to be happy and enjoy their visit ("*Enjoy visit*") in order to keep (or convert) them as loyal (recurrent) visitors, he has to show him beautiful eye catching and attractive things about what he's looking for ("*Provide multimedia material*"), and by giving visitors the opportunity and the hope to win a prize, the website owner transform their visit in an emotional experience which they'll keep a memory of and in an additional reason to come back visiting the website.

Updates

Some visitors which either just want to stay updated or are waiting for news about particular subjects ("*Stay informed of updates*") want to subscribe to news in order to be alerted as soon as something that might interest them is published. That's why the website owner provides newsletters and news feeds. This is also two way relationship as each subscriber is a potential recurring visitor that accepts to be reminded of the website and invited to new visits on a regular basis.

10.2.4 Partner websites

Nearly all websites have a similar goal : being linked by other websites of quality in their domain. The websites of the product line are no exception and collaborate with other websites to reach these goals.

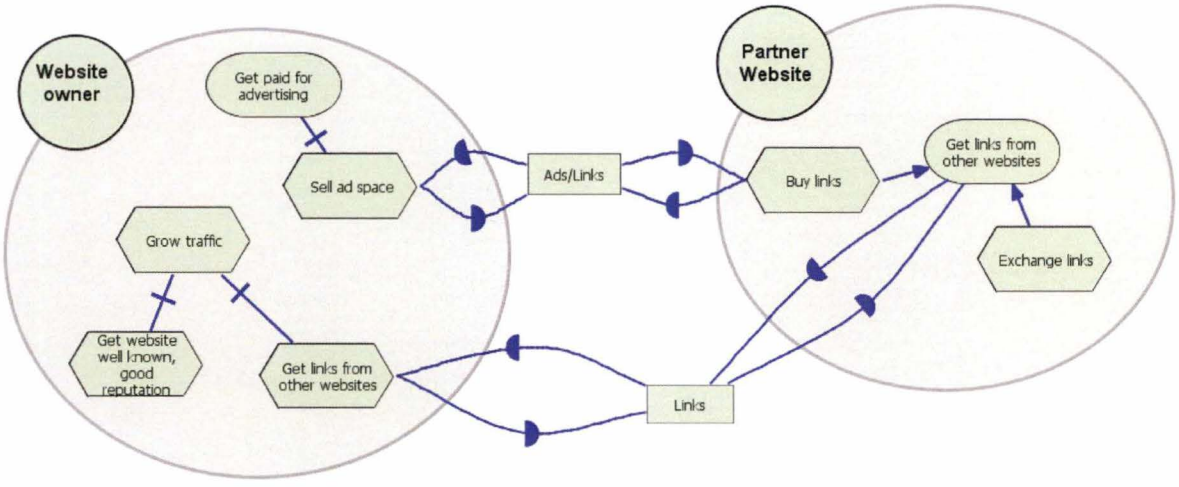


Figure 10.5: Goals related to partner websites

As mentioned before, one of the goals of all websites is to get links from other websites ("*Get links from other websites*").
The reasons webmasters want to get links from other websites are few and simple :

- Links drive new visitors
- Links from other websites improve the site's reputation and thus rankings in search engines results, which in turn drive new visitors

Getting linked by other websites occur by :

- Exchanging links with other websites ("link me and I'll link you back")
- Buying links or advertising space against money
- Other webmasters know your website and find it interesting for their visitors and decide to link you

10.2.5 Distribution companies

Distribution companies advertise in a special way. Rather than buying advertising space, they encourage the websites to talk about their products.

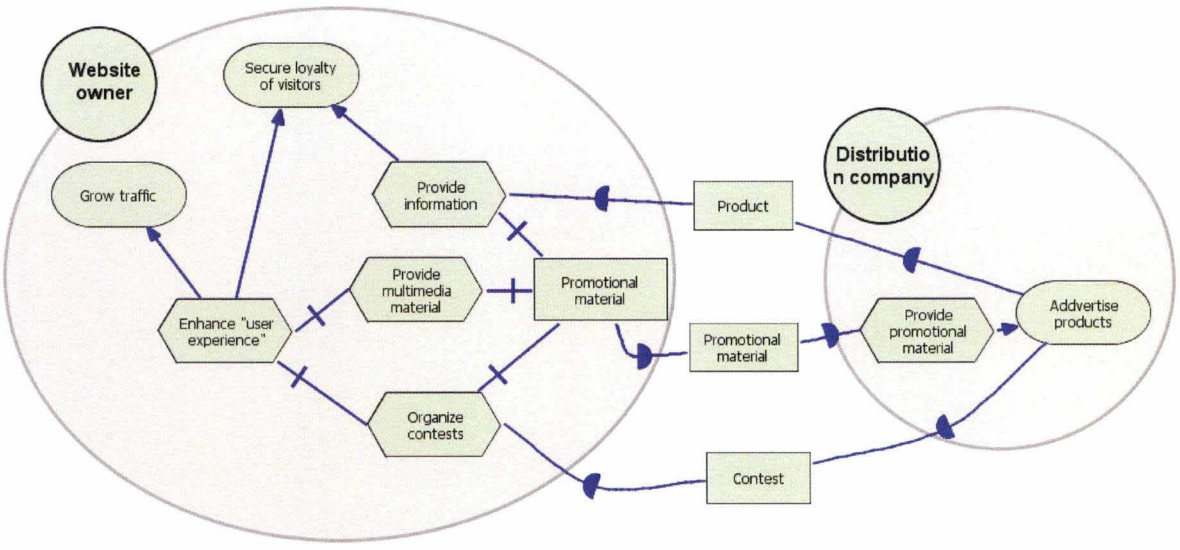


Figure 10.6: Goals related to distribution companies

Distribution companies encourage the website owner to talk about their products in various ways :

- They provide promotional material ("*Provide promotional material*") that can be displayed on the website if it is digital material (videos, wallpapers, etc.) ("*Provide multimedia material*") or that can be offered as prizes (t-shirts, dvds, ...) for the winners of contests ("*Organize contests*").
- They provide the website owner with products for free and asks him to publish reviews about them ("*provide information*").

10.3 Scope and requirements

10.3.1 Planned products

Six products are planned to be grouped as a product line :

- Film-Critik
  This website publishes movie reviews and other information about movies, like the actors that played in a given movie, the people that directed it, the company that distributes it, the year it was released, the country it is from, ...
- Actors / Directors
  Information about actors and directors of movies. Biography, pictures, quotations, ...
- Books : reviews and information about books
- Authors : information about authors
- Soundtracks : Original soundtracks made available online
- Musicians / Bands : Information about musicians and bands

Most of these websites are very similar in structure and features. For example, the website about books is very similar to "Film-Critik" about movies.

10.3.2 Identification of features

In order to identify the features, a rough "requirements analysis" has been conducted for each website that is foreseen as a product of the product line. This requirements analysis merely consists of a bunch of sketches describing the websites and the different pages they must contain and their content.

The sketches availaible in A show clearly that web pages from all products share a common standard structure containing basic elements that are always present. The following schema (figure 10.7) represents them as a feature diagram.

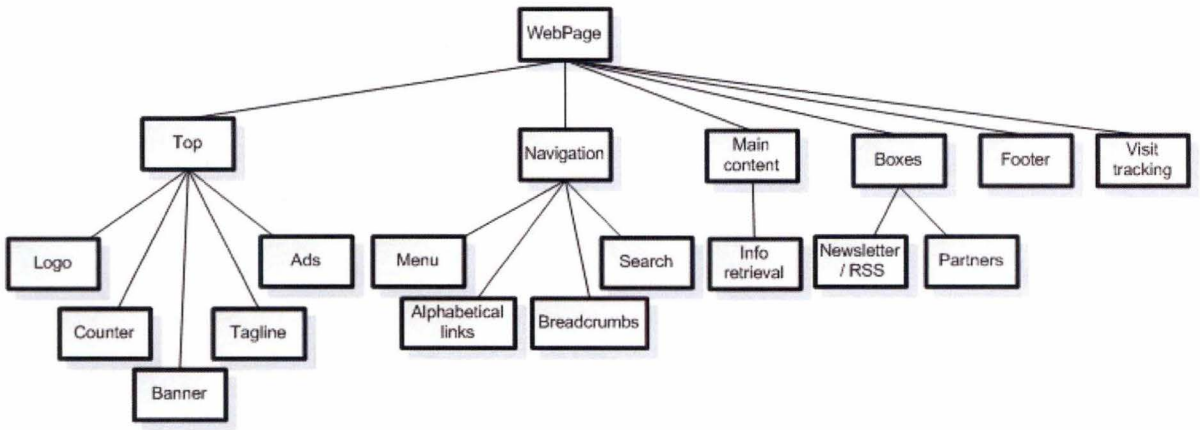


Figure 10.7: Common mandatory features

Of course most of these features need to be configured or even tailored specifically for each product (in order to keep the schema simple and easy to read, attributes are not shown, but will be described further) but we can already see that a lot of commonalities exist across the different products, and that each product can be derived from a single common model.

Description

- Menu :
  The menu is a simple and very classical navigation control which presents the main parts, the important entry points in the site and allows the visitor to browse from one to another. It is always displayed on all pages of the website.

- Tagline :
  The tagline summarizes the content of the website in a few words. It serves as an information as well for the visitors who can immediately understand what the website is about when they read it, as for the search engines. It is always displayed on all pages of the site, often somewhere near the top of the page and is a link to the homepage.

- Counter :
  Each website presents information about a specific subject and has one page for each 'instance' of the main subject, the counter displays the current number of instances of the main subject the website has information about.

- Ads :
  In order to generate an income, the website rents advertising space. Each time a visitor clicks on an advertising, the announcer is charged and a part of the fee goes back to the website owner. The possibility exists for various types and sizes of advertising spaces, and it depends on the website. The previous versions of

film-critik only present 2 or 3 small ad blocks, one at the top of the page near the banner, and two within the text of the page. There are several advertising services available on the web. They often give a small piece of code to include in the page at the place where the ad must be displayed.

- Footer :
  The footer discretely summarizes a few pieces of information (like copyrights) and links to pages that are not really important to the user's experience but that need to be presented on every page.

- Banner :
  The banner is like a graphical version of the tagline. It is presented near the top of the webpage and often contains an illustration and / or a short text, a quote or something that makes the visitor want to stay and read more. It is often a purely static element of the page that never changes but some websites have dynamic banners that can change on basis of whatever the designer may have decided.

- Visit tracking :
  The traffic of a website in terms of visitors, visits and a few other key numbers is a reference metric to measure the success and the popularity of a website. The visit tracking services offer really valuable information to the website owners, which can use this information to improve the quality of the site and as a consequence the visitor's experience and the popularity of the website.

- Feed :
  Syndication feeds, more commonly named RSS feeds (although there are other formats like ATOM) are a simple and powerful way to inform the visitors that are interested in your updates that something new is available. The concept behind this is rather simple : the visitor subscribes (using a feed reader) to the feed of a website, and each time something new is available, the feed is updated so that the visitor know something new is available. Some feeds are full while others are partial. Full feeds contain the full text of what has been added to the website while partial feeds contain only an abstract and a link to full version of the page located on the website.

- Search :
  A search box is present on each website to allow the visitor to searchin the content of the whole website without having to go through menus and lists to find what he's interested in. This search box can be provided by different well known search engines

- Home page :
  The home page is not actually a feature in itself but it is an element present in every website. The content of the homepage is really specific to each product and can completely vary from one website to an other. It presents the content that the visitor must see when he browses the "home" page of the site. That is, the page he sees when the url he uses only consists of the domain name and no specific page is requested.

- Contact form :
  The contact form allows the visitor to send a message to the website administrators. It is a fairly common feature. Each website of the product line will have its contact form.

- Info pages :
  The information pages are the core of the website, this is where the visitor will find the relevant information he's looking for. Their content will really be specific to each product. Just like other pages, they can be surrounded by boxes containing other information.

- Automatic links :
  The automatic links is a feature that will allow to display a piece of information about an instance of the main

subject of the site as a link to an other page specialized on this particuliar piece of information. For example if an actor is listed in the casting of a film and that a specific page about this actor exists in the website about actors, his name will be transformed into a link to the actor's page on the actors website. However, if no page exist for a given actor, his name is written as plain text.

- News :
  The news system is a more complex feature, it involves a component that displays the last news, a form to write news, and a kind of news generator that will create posts when certain events occur on the website (e.g. when new information is added).

Optional features

Some pages may contain other features that are not available on others. The following diagram in figure 10.8 describes it.

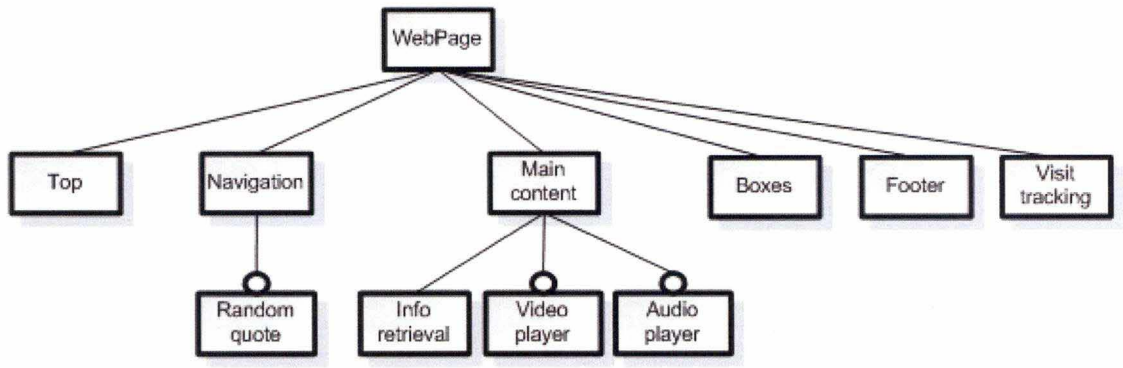


Figure 10.8: Optional features

- Random quote :
  A random quotation is extracted from the database and displayed with a link to the page about the work in which it appears.

- Audio / Video player
  Audio and video players can be embedded to play sound tracks and videos obtained from different external sources.

Boxes

The main content of a webpage is surrounded by a bunch of small "boxes". These can contain various type of content :

- Static content :
  Purely static content that never (or almost never) changes, examples of this type of boxes are the links to the newsletter and the rss feed and the box containing the links to the partner websites.

- Dynamic content :
  Boxes with dynamic content can virtually change continuously. These boxes are often based on a db query from which the result is properly formatted to fit the size and shape of the box. Examples of this type of boxes are "Currently in cinemas" and "Last added".

- Semi dynamic content :
  Boxes with semi-dynamic content are between static and dynamic content, they vary from page to page, but for a given page, the content never (or almost never) changes. The best example of boxes of this type is "Related".

Classifying boxes in these categories will be useful when defining attributes as each category will determine the configuration parameters needed by the boxes that belong to it.

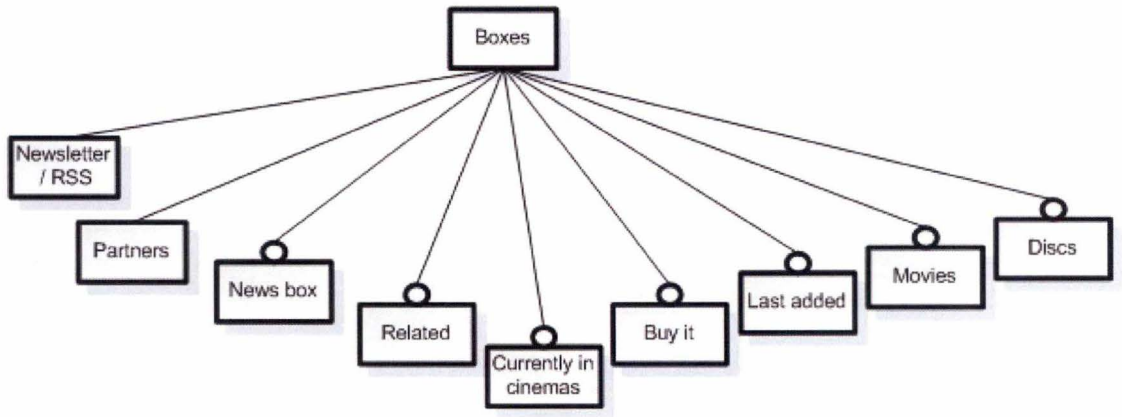


Figure 10.9: Boxes

The screenshot in figure 10.10 maps the feature names to their actual presentation on the website.

Figure 10.10: Features

However, products are not only web pages, they are web sites, containing, among other things web pages but also some other features. The diagram in figure 10.11 describes the features that are common to all products of the product line.

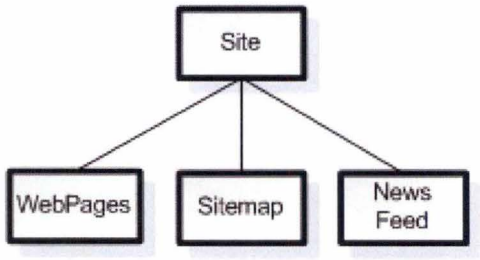


Figure 10.11: Features common to all web sites

- Webpages :
  Webpages were described more in details previously. They contain information about a subject.

- Sitemap :
  Sitemaps are a great way of letting the search engines know about the pages your website contains. A sitemap is a simple XML document formatted following a given standard that (nearly) all search engines understand. It can be useful when some pages are difficult to reach through the navigation controls or when search engines can not use the navigation controls.

- News feed :
  The news feed is a simple XML document that is updated each time news are published. It is fetched regularly by feed readers and aggregators which can use its content to display the latest news externally, somewhere a user wants to see them, in the form he wants to see them.

10.4 Putting goals and features together

The table 10.1 shows to which goals each feature contributes. It shows that every goal or main task is handled by one or more features and that no feature is useless.

	Enhance user experience	Secure loyalty	Grow Traffic	Get paid	Provide updates	Grow subscriber base	Provide navigation	Provide multimedia	Provide info	Get links
Counter	X									
Tagline	X									
Advertising				X						
Banner	X									
News Feed	X				X	X				
Sitemap			X							
Search	X						X			
Contact form	X									
Video player	X							X		
Audio player	X							X		
Menu							X			
Tracking		X								
Logo										
Alpha links							X			
Breadcrumbs							X			
Info retrieval									X	
Newsletter					X	X				
Boxes		X	X	X					X	X

Table 10.1: Features x Goals

10.5 Organizational context

- There is currently a single website providing a part of the features planned to be in the scope of the product line but not all of them.
- The new version was planned to make use of a new data structure that is not compatible with the existing code, so that most of the existing assets will have to be completely rewritten from scratch to work with the new data structure. (See 10.6.3ARC).
- The implementation of features may vary among the products. Some features are really product specific while others can be reused "as-is" across all products.

- New similar products are very likely to be added in the future.

10.6 Technology forecasting

10.6.1 Php / MySQL

The first version of film-critik was completely written from scratch using php as scripting language and mysql as database engine. These technologies were chosen because :

- They are very common, well documented and supported by the user community
- The team already had the required knowledge and some experience in building websites using these technologies
- Php/MySQL hostings are cheap and easy to find

At the very beggining, the team had no idea about the traffic the website would drive nor about its lifetime, so even though the team had the knowledge and some previous experience in building websites using other technologies like Java or .net, the lightweight php / mysql solution was preferred.

10.6.2 ModX

Starting from the second version of film-critik, the decision was taken to make use of ModX, an open source content management framework that offers a strong basis to build new features upon.

Several reasons lead to the use of ModX :

- It's open source and well documented, and as such, it is easily tweakable and customized.
- The framework and the API offer a lot of services that can be used and that don't need to be developed anymore. Among them we can mention :
  - User management, authentication and security
  - Template engine and caching
  - Document parser and error handling
  - Database access
- In addition to the services included in the framework, a lot of plugins and code snippets are readily available (some of them are even included in the basis package). The user comment form, the contact form, the login form, the news aggregator and the "breadcrumbs" are a few examples of these.
- It offers a simple way to develop and integrate new plugins and code snippets.

Besides these technical motives, a few other reasons weighted in favor of ModX :

- It's free
- It's php/mysql based, the first versions of the site were already php/mysql based and there was no plan to change the hosting.
- The team already had the knowledge needed to work with ModX before starting.

The team already knew about other CMS that could have been used as a basis to build the website but none of them was as easily extendable and customizable. ModX is a true content management framework built to be extended.

10.6.3 ARC

ARC is the framework that will be used to access the "business" data that provides the information presented on the websites. It allows to query external SPARQL endpoints, store data as triples, query the data store using the SPARQL language, parse and generate RDF documents and even expose a public SPARQL endpoint.

ARC is particularly well suited to act as a data layer in the product line thanks to its special data structures that avoid any specific database design. ARC stores everything as triples (object - relation - subject) and has its own set of standard tables that do not change, no matter what information is stored. This means that every product will be able to use the exact same set of tables to store and retrieve the information it needs. As such, the ARC framework will be used as a core asset that will be part of the product line architecture.

10.7 Summary

In this chapter we have covered the context in which we will develop our products. We have seen that all the products are related to the cinema and other arts that can are related to it. We have described the goals that we want to reach, on one side the goals of the websites, and on the other side, those that we aim for by introducing a product line approach. These goals are very common, growing traffic and being paid for advertising are goals of many websites, and the expected benefits of the approach are commonly reported as reasons to build a product line.

We have described the various features that will be provided by the websites and observed that a great part of them are common to all products, even if some of them are optional. The constraints described in sections 10.5 and 10.6 are mostly technological (use of php/mysql and frameworks).

The next chapter will describe how all these elements will be decomposed and assembled together to ensure that the system offers the wanted variability and that components will be reusable.

Chapter 11

Establish production capability

This chapter describes the elaboration of the core assets that will also be the core of the product line. These assets are the building blocks with which everything will be built. We will first describe the architecture which is the common basis for all products that are being developed (section 11.1). This is where all variation points will be described and where we will determine how variability will actually be implemented for each of them. Then in section 11.2, the link will be made between features and the assets that will implement them. The next sections (11.3 and 11.4) will describe which features will be included and how assets are going to be be assembled together in order to form the desired products. Then, in section 11.5we will see which tools can be used to automate the production processes.

11.1 Architecture

11.1.1 Decomposition and layered view

The figure 11.1 describes visually the layers of the architecture that serves as basis for the products.

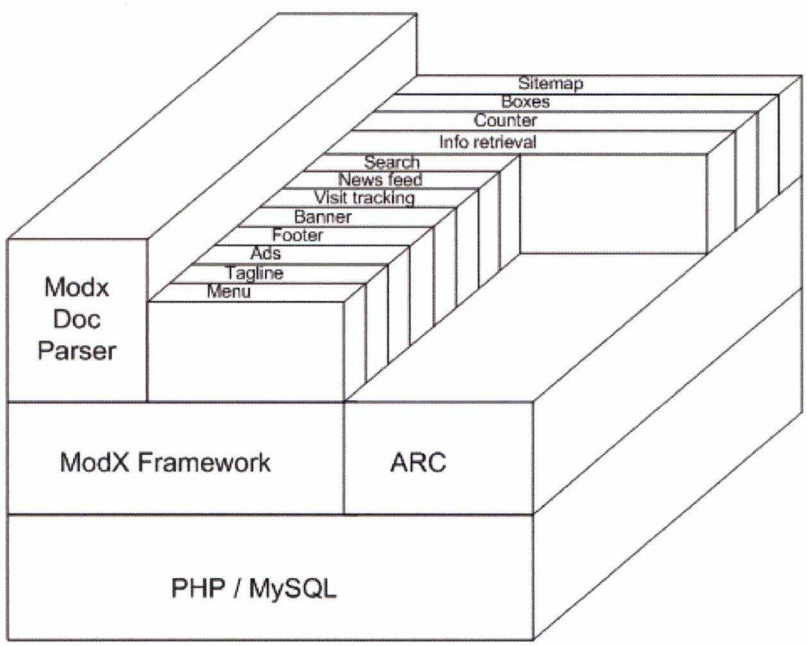


Figure 11.1: Architecture layers

Php / MySQL

As the products are all based on Php as programming language and MySQL as database engine, these technologies are naturally the most basic layer of the architecture. That's what everything relies upon.

ModX Framework

The ModX framework is used as basis for every component. As described in 10.6.2, it offers a lot of services that are used. The database access API it offers is probably the most simple and used of all services. It should also be mentioned that whatever the component does, it has to be bound to ModX in the ModX manager, that's why they all rely upon it.

ARC

As described previously in 10.6.3, ARC is the framework used to access "business" data in a simple and unique manner, no matter what relationships may bind the different data together. It is required by every component that accesses these "business" data, the most important of all being "Info retrieval". Some boxes use ARC, while other don't. Detailing every type of box and placing it accurately on the schema would overload it and make it unreadable.

Modx Document Parser

The ModX Document Parser is an important part of ModX. It is what parses the documents fill the templates with their content, find chunks and snippets, execute them and place the result at the right place, execute the plugins and last but not least, returns the result to the visitor. It is the component that interfaces the system with the visitors. It relies intensively on the ModX framework to find and read or execute the components that implement the features (Menu, Tagline, Ads, ...).

11.1.2 Interaction with other systems

- The syndication feeds use standard formats (RSS/ATOM) to ensure compatibility with feed readers
- The sitemaps use the standard sitemap format to ensure most of the bots can handle it.

11.1.3 Description of variation points

- Counter
  - What varies ? The type of objects being counted and the description
  - Variability realization : At build time, the type of objects is replaced in the snippet by the value found in the configuration and a chunk is created with the given description.
  - Configuration
    * Type of objects to count
    * Description of the type of objects to display
- Tagline
  - What varies ? The text displayed on the webpage
  - Variability realization : At build time, the placeholder for the tagline contained in the templates is replaced by the content of the file selected in the configuration.
  - Configuration
    * Name of the file containing the tagline

- Ads
  - What varies ? The code being used to display the ad.
  - Variability realization : At build time, the chunks are created using the names and files given in the configuration.
    * Names of the chunks
    * Files containing the code to be included in each chunk
- Footer
  - What varies ? The text displayed on the webpage
  - Variability realization : At build time, the placeholder for the footer contained at the bottom of the templates is replaced by the content of the file selected in the configuration.
  - Configuration
    * Name of the file containing the footer
- Banner
  - What varies ? The image being used.
  - Variability realization : A simple replacement of the file that contains the image for the banner.
- Tracking
  - What varies ? The code being used to call the tracking service.
  - Variability realization : At build time, the chunk is created using the code contained in the file specified in the configuration.
  - Configuration
    * Name of the file containing the code for the tracking.
- Feed
  - What varies ? Nothing.
- Sitemap
  - What varies ? The type of elements being included in the sitemap.
  - Variability realization : At build time, the type of objects is replaced in the snippet by the value found in the configuration.
  - Configuration
    * The types of the objects for which a page is created and that must be included in the sitemap.
- Search engine
  - What varies ? The code being used to use the search service
  - Variability realization : At build time, the chunk is created using the code contained in the file specified in the configuration.
  - Configuration

    * Name of the file containing the code for the search engine.

- Boxes
  - What varies ? The boxes being displayed around the different pages
  - Variability realization :
    * Boxes : A list of boxes described by a title and a snippet name, and a column are contained in the configuration. For each box, a standard box is derived from a generic box template, the title is replaced and the snippet is called in the body of the box, then the resulting code is placed in the given column in the template.
- Info retrieval
  - What varies ? The type of objects for which information is being retrieved
  - Variability realization : At build time, the type of objects is replaced in the snippet by the value found in the configuration.
  - Configuration
    * Type of objects from which information is to be retrieved.

11.2 Products parts and assets

11.2.1 Preexisting assets

ModX

ModX is the framework upon which everything relies. It is a central master piece of the product line architecture.

Standard ModX snippets

- eForm
  eForm provides a simple yet efficient way to create a contact form (with or without captcha check) through which the visitor can send a message to the website administrators.
- Breadcrumbs
  breadcrumbs automatically generates a trail of pages through which the current page can be accessed and allows to go back to any upper level in the website hierarchical structure.
- Wayfinder
  Wayfinder is a menu builder. It automatically builds a menu from a subset of the documents contained in the ModX tree.
- WebLogin
  Weblogin allows a visitor to log in in order to gain access to protected pages and displays a logout link when the visitor is already logged.

ARC

ARC is the framework used to access data.

SPARQL Endpoints

Besides the original content provided in the websites (mostly reviews and comments), a lot of pieces of general information are also made available. These pieces of information are available almost anywhere on the web and could be imported manually using a data entry form, but can also be fetched in a structured form from endpoints that generate RDF formatted data. These endpoints can be considered as assets that will provide data when needed.

Data from previous versions

The first product (film-critik.net) will benefit from the data that were produced during the lifetime of the previous versions. All the reviews and comments about the movies that already exist are being imported and used. Reusing these data will have a cost : they are stored in specific database tables and will need to be converted to be stored using the ARC framework.

11.2.2 Product parts

Counter

The counter is made of a snippet (instanceCounter) which executes the query on the database and replaces the placeholder located in a chunk named "instanceCounter" by the result of the query and returns the result of the substitution.

Tagline

The tagline does not have any real physical asset, as it is the result of a replacement in the templates.

Advertising

The ad blocks are contained in chunks (ad-banner and ad-content) which contain the code, respectively for the ad block located in the banner and for the blocks contained in the content of the page.

Footer

The footer does not have any real physical asset, as it is the result of a replacement in the templates.

Banner

The banner is only made of an image file (banner.jpg)

Tracking

The visitor tracking system is made of a simple chunk (tracking)

Feed

The news feed is made of a special ModX document, just like the sitemap, which contains a call the "newsfeed" snippet.

Sitemap The sitemap is made of a special ModX document (type XML, named "sitemap") and a snippet (sitemap) called from within the document

Search engine

The search engine is made of a simple chunk named "search"

Boxes All pages contain "boxes", that is, small parts that display a little piece of information. Boxes often contain short lists of items, but they could contain almost anything. The content of each box is generated in the same way. A specific snippet executes a database request and then formats the results using a template contained in a chunk.

11.3 Planned products

Six products are planned to be grouped as a product line :

- Film-Critik
  This website publishes movie reviews and other information about movies, like the actors that played in a given movie, the people that directed it, the company that distributes it, the year it was released, the country it is from, ...
- Actors / Directors
  This website publishes information about actors and directors of movies. These information contains biography, pictures, quotations, prizes won,...
- Books
  This website publishes book reviews and other information about books (author, edition ...)
- Authors
  This website is almost identical to "Actors / directors" except that the information published is about authors of books, novels, scenarii, ...
- Musicians / Bands
  Just like "Authors", this website is almost identical to "Actors / Directors" except that the information provided relates to musicians and their bands. These information includes biographies, discographies, links to their website, music they made, ...
- Soundtracks
  This website embeds external multimedia content using the audio and video players to group together audio and video material related to a movie or a musician.

Most of these websites are very similar in structure and features.

11.4 Production plan

11.4.1 Inputs needed

There are a small number of inputs needed to build a product :

- A working php/mysql hosting
- The core software assets base
- A description of the product mentioning :
  - The variants of the mandatory features to be included in the product
  - The optional features that are to be included and the variants to be used.
  - The configuration information for every feature or asset that requires it.

  This description can be generated using a configuration tool that would allow the user to select and configure the various features and assets that are to be included in the product being built.

11.4.2 Assumptions

- There is an automatic integration tool that knows how to handle a product description in a given (formal) format.
- Every software asset that is required by the product description is included in the core software asset base

11.4.3 Detailed production process

The activities to perform to yield a complete product are the following :

Hosting creation and configuration

Before anything else can be done, an hosting has to be created on a server that allows php to be used in conjunction with a MySQL database. A database must be created as well.

An ftp account must be defined with appropriate permissions to allow to upload files.

A database account must be defined with sufficient rights to create tables (CREATE) modify them (ALTER) and use them (INSERT, UPDATE, DELETE, SELECT).

ModX CMF Installation

The ModX CMF must be installed on the hosting before any product can be created, as it is the basis on which every product relies heavily.

ARC Installation

The ARC framework must be installed as it is the data access framework that will be used by every product.

Architecture installation

Every asset required by the architecture of the product must be installed.

Product specific assets installation

Additional assets required by the product in the product description must be installed. Which assets are actually needed is described in or derived from the product description.

External tools configuration

Each and every external tool that is used by the product should be configured properly.

Currently these products are :

- Google Analytics and Php My Visites : These tools are used to track visitors and report statistics about the website usage.
- Google Webmaster tools : This tools are used to get various information about the website and its environment, external (incoming) and internal links, potential problems (page structure, page not found, ...), how search engines see the website, ...
- Feedburner : This tool serves a a proxy for the news feed to which it adds services like automatic pinging, addition of advertising within the feed, usage statistics, ...

11.4.4 Production strategy

The product line will be built using an incremental approach. This means that, at first, only the set of core assets needed for the first product will be developed. Then, when the first product is ready, only the new assets needed for the second product will be developed, and so on.

11.5 Tools

Although the production process can not be entirely automated, tools can take care of some steps, namely the installation of the architecture and of the assets, which are the most tedious and error-prone steps of all.

The use of an automated tool to put the various core assets together implies that each product will need its own formal description defining which assets it requires and each of these assets will also need a formal description of its structure, its installation process and its dependencies. To build this formal description means to take decisions for the choices left open at variation points. A configuration tool can help to select options and maintaining consistency by preventing to make choices that would break the rules imposed by the feature model.

11.5.1 Configuration tool

QConf has been developed to be a graphical configurator for the linux kernel. It provides a graphical interface which allows to select a valid set of features by presenting available options and maintaining the selection coherent with the constraints that might exist between features. The available features and the constraints are described in a project file that uses a simple syntax (see 11.5.1). Once the selection has been made, it can be stored in a file. A command line version exists if needed.

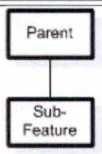
Features

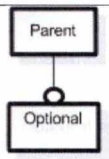
- It presents variants graphically
- It verifies constraints
- It allows parameters to be configured as text, integer or hexadecimal values.
- It can easily be customized, the project file can even be generated from a feature diagram if a formal format exists.
- It generates a configuration file in a very simple format, easy to use by other tools
- It is free, open source and readily available

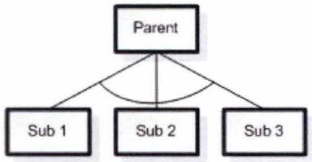
It produces a file containing the values selected during the configuration process.

LKC Language This paragraph presents the syntax used to define a feature model in QConf and the corresponding notation in feature diagrams. In (SW), the authors propose such a mapping, the one presented here presents the same concepts using the syntax defined more formally by Roman Zippel in (Rom02).

Type	Feature Model	LKC syntax

Mandatory		`config Parent`
		`  boolean 'Parent'`
		`  select Sub`
		
		`  config Sub`
		`  boolean 'Sub'`

Optional		`config Parent`
		`  boolean 'Parent'`
		
		`  config Optional`
		`  boolean 'Optional'`
		`  depends on Parent`

Or		`menu P`
		`  config Sub1`
		`  boolean 'Sub1'`
		
		`  config Sub2`
		`  boolean 'Sub2'`
		
		`  config Sub3`
		`  boolean 'Sub3'`
		`endmenu`

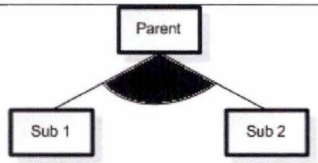
Xor		`choice`
		`  prompt 'Parent'`
		
		`  config Sub1`
		`  boolean 'Sub1'`
		`  config Sub2`
		`  boolean 'Sub2'`
		`endchoice`

Table 11.1: Feature Model to LKC mapping

Graphical user interface

As the figure 11.2 shows it, QConf offers a simple yet powerful graphical user interface to select features.

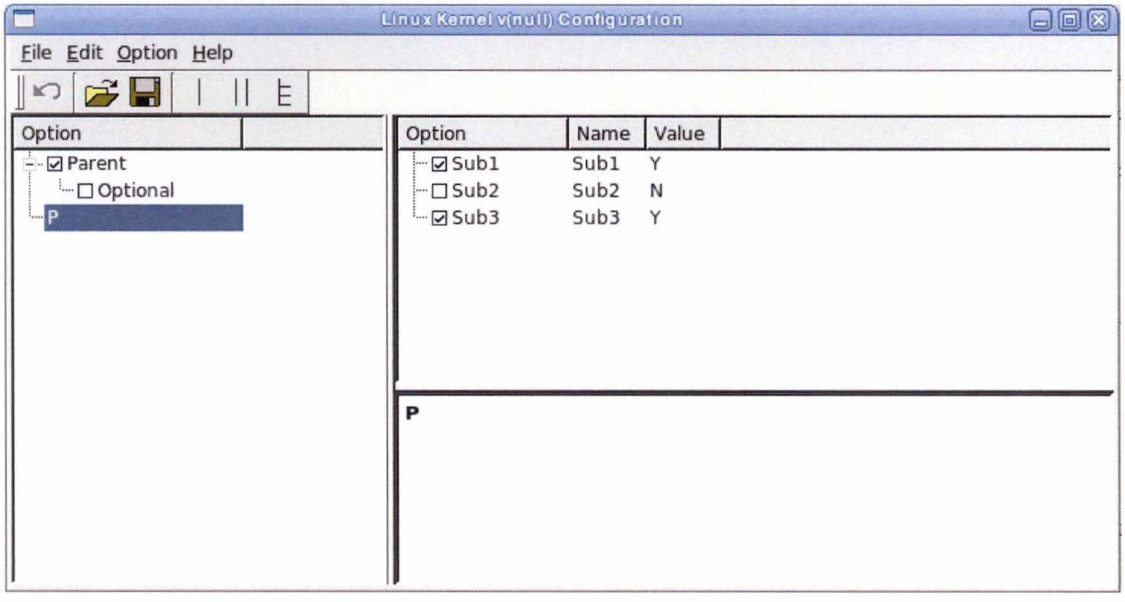


Figure 11.2: QConf screenshot

11.5.2 Integration tool

Based on the feature set contained in the file generated by QConf, and a formal description of the relationships between features and the "physical" core assets, a specific tool can generate products automatically. Such a tool must be developed specifically.

11.6 Summary

In this chapter, we have described the core asset base that will be used to build the products. In section 11.1, we have described the architecture and we can see that it is quite simple thanks to the extensibility offered by the ModX framework. The analysis of the variation points shows that even if many features are mandatory and common to all products, most of them vary nevertheless. The associated variation points are often of type "value" and must be tailored for each product by defining some parameters. In section 11.2, the assets that constitute the features are described, and we can notice that nearly all of them are implemented using snippets and chunks or physical files in the case of images. The description of the products in section 11.3is a bit perfunctory, but more detailed sketches of the products are available in appendix A and a formal description will be generated by a configuration tool. In section 11.4, the production plan details the actions to accomplish to build a product as well as everything that is needed to do so. Finally the section 11.5 describes the tools that will be used to automate the production process.

Part IV

Evaluation & Conclusion

Chapter 12

Evaluation of the methodology

The chapters 8 and 9 presented and detailed the methodology that will be used in the rest of this work. This chapter is intended to review it with a critical mind to identify qualities and shortcomings and propose improvements.

12.1 General structure

The methodology presented in the previous chapters is centred on software engineering and voluntarily ignores organizational and management aspects except for the organizational context that allows to define some organizational constraints, but it does not provide any advice or guidelines on these subjects.

It reuses the traditional split between analysis of context and applications that is the basis of many other methodologies and focuses on the development of core assets and products. It is clearly only intended to be used during the migration between a single software development and a software product line, for it does not describe how to run the product line nor how to add new products, features and assets to the product line.

12.2 Context

All the important aspects of the context of a product line are aborded and described, however it should probably be mentioned that the analysis of the goals is not defined by the "SEI Framework", and is therefore a custom addition. Analyzing the goals of the products and of the product line is not mandatory nor essential but it brings some advantages like a better understanding of why products are built and what matters most. Designing goal models is also useful to explicit the use of features, ensure that the features in the scope will suffice to satisfy all the goals and that no useless feature is developed.

A section about imposed development standards could be useful in certain situations but as there weren't any special constraints of this type in our case, there is also no reason to add it. Technological standards are specified in the "Technology forecasting" section.

12.3 Production capability

Again here, all the important artifacts and assets needed to actually build products are mentioned from the most basic layer, the architecture, to the tools that will help to integrate the different products parts in the end.

This part of the methodology mostly describes what has to be done and describes some qualities of the desired result but does not give any direction, advice or guidelines on how to achieve them. Some vague qualities of the architecture and the reusable components are mentioned but no design principles or patterns are presented as basis to build them correctly. Some parameters are defined for the choice of the variability realization technique but the

method does not detail how to select a technique from the answers given to the proposed questions. However, a methodology could only present examples and propose solutions but certainly not impose them, for there are numerous possible realization techniques and the choice of the realization technique also depends on the technologies involved. The section about the production plan describes all the important points of what should be detailed in it, but it could go a little further and propose some questions that should be answered in order to make the production process clearer and easier to execute later. The same remark applies to the choice of the tools, where the type of tools are detailed but not the expected qualities nor a way to evaluate the adequation of a tool.

Another potential improvement would be the addition of a section that would deal with the testing of the software produced and how tests are being reused across products.

12.4 Summary

This chapter analyzes and present a review of the methodology presented in the chapters 8 and 9. Besides the fact that the methodology is globally good and even adds an interesting addition to what is prescribed by the framework, a few potential improvements have been identified. We have seen that it has been specifically tailored to the development of a product line based on the Film-Critik website and that every irrelevant point has been omitted. We have also constated that the method mostly focuses on what must to be done and not much on how to do it.

Conclusion

Summary of contributions

This work has shown through a pratical example how using a software product line approach can allow to develop several products (and open doors to build even more) starting from almost nothing. In this case, a first product existed before and served as example and basis to draw the lines of the new products without changing really much, but everything could also have been imaginated from scratch.

We have first reviewed various existing methods and frameworks to build product lines, then derived a custom method from the framework of the Carnegie Mellon University, we have described it and applied to our case study.

Analyzing commonalities and variabilities of the different products has lead to build an architecture which serves as a basis for every new product that will be supported by the product line, and define where and when the different products vary as well as how to support these variations. The reusable components that implement the different features of the products were built, and the process to assemble them was described. Tools are used to simplify the configuration and integration of these components to make end-products.

The method that was developed and used was then analyzed and reviewed for potential improvements.

The websites described in this case were kept simple, just as were the problems and the solutions. This has advantages and disadvantages.

The biggest disadvantage is probably that the product line described here is not as developed as a product line that would have been developed in the context of a real big enterprise. Important concepts of the product line have been greatly simplified or completely omitted, like the management aspects for example.

Some concepts and techniques have not been treated in depth and should deserve a dedicated work on their own.

On the other side, keeping things simple has allowed to discover easily the concepts and the processes that were presented.

Costs and benefits

Although the analysis and development of truly reusable components has taken much more time that it would have for a single product, adopting a product line approach was totally appropriate when considering the cloud of websites that will revolve around Film-Critik.net. Besides the advantages (in terms of time, simplicity and quality) that this approach will bring when building the new websites, it will also allow to release new versions of existing websites more often and more easily.

Perspectives

This work is was an introduction to software product lines with a pratical example. It is certainly not complete. A lot of things can be done to go further than what is described in this document. Some components that are part of this product line (like boxes for example) could also be products of a product line of components, which in fact

means that variabilities could have been analyzed deeper. Templates could also be products of another product line before being integrated in this one.

Bibliography

[AHYG07] Samer Abdulhadi, Jennifer Horkoff, Eric Yu, and Grau Gemma. i* guide, 2007. Available from World Wide Web: `http://istar.rwth-aachen.de/tiki-index.php?page_ref_id=67`.

[BB01] Felix Bachmann and Len Bass. Managing variability in software architectures. *SIGSOFT Softw. Eng. Notes*, 26(3):126–132, 2001. Available from World Wide Web: `http://dx.doi.org/10.1145/379377.375274`.

[Ber07] Thorsten Berger. Softwareproduktlinien entwitcklung - domain engineering: Konzepte probleme und loesungansatze. Master's thesis, Univeristat Leipzig, Fakultat fur Mathematik und Informatik, Institut fur Informatik, 2007.

[BFK+99] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. Pulse: A methodology to develop software product lines. In *SSR*, pages 122–131, 1999. Available from World Wide Web: `http://dblp.uni-trier.de/db/conf/ssr/ssr99.html#BayerFKLMSWD99`.

[Bos] Jan Bosch. Software variability management. Available from World Wide Web: `http://www.janbosch.com/01SVM-Introduction.pdf`.

[Bos00] Jan Bosch. *Design and use of software architectures: adopting and evolving a product-line approach.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Bos04] Jan Bosch. On the development of software product-family components. In *SPLC*, pages 146–164, 2004.

[CB05] Paul Clements and Felix Bachmann. Variability in software product lines (cmu/sei-2005-tr-012). Technical report, Software Engineering Institute, Carnegie Mellon University, 2005.

[CM02] Gary Chastek and J. D. McGregor. Guidelines for developing a product line production plan. Technical Report CMU/SEI-2002-TR-006, Software Engineering Institute, 2002.

[CN01] Paul Clements and Linda Northrop. *Software Product Lines : Practices and Patterns*. Addison-Wesley Professional, August 2001.

[DSNB04] Sybren Deelstra, Marco Sinnema, Jos Nijhuis, and Jan Bosch. Cosvam: A technique for assessing software variability in software product families. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 458–462, Washington, DC, USA, 2004. IEEE Computer Society.

[Har02] Maarit Harsu. Fast product-line architecture process. Technical Report 29, Institute of Software Systems, Tampere University of Technology, January 2002.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, 1990.

[KMB06] Vitaly Kudsidman and David M. Bridgeland. A classification framework for software reuse. *JOT : Journal of Object Technology*, July 2006. Available from World Wide Web: `http://www.jot.fm/issues/issue_2006_07/article1/`.

[Mat04] Mari Matinlassi. Comparison of software product line architecture design methods: Copa, fast, form, kobra and qada. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 127–136, Washington, DC, USA, 2004. IEEE Computer Society.

[NCB+09] Linda M. Northrop, Paul C. Clements, Felix Bachmann, John Bergey, Gary Chastek, Sholom Cohen, Patrick Donohoe, Lawrence Jones, Robert Krut, Reed Little, John McGregor, and Liam O'Brien. A framework for software product line practice, version 5.0, 2009. Available from World Wide Web: `http://www.sei.cmu.edu/productlines/framework.html`.

[Pol06] Thomas Pole. Software reuse: History 1980 to 2005, 2006. Available from World Wide Web: `http://www.computertrainingguild.org/JHU/SWReuse_Spring06/L01%20Software%20Reuse%20History.ppt`.

[Pou96] Jeffrey S. Poulin. *Measuring software reuse: principles, practices, and economic models*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[Rom02] Zippel Roman. alternative linux configurator specification v0.1, March 2002. Available from World Wide Web: `http://lkml.indiana.edu/hypermail/linux/kernel/0203.2/1376.html`.

[Sch99] Douglas C. Schmidt. Why software reuse has failed and how to make it work for you, January 1999. Available from World Wide Web: `http://www.cse.wustl.edu/~schmidt/reuse-lessons.html`.

[SvGB05] Mikael Svahnberg, Jilles van Gurp, and Jan Bosch. A taxonomy of variability realization techniques: Research articles. *Softw. Pract. Exper.*, 35(8):705–754, 2005.

[SW] Julio Sincero and Schroder-Preikschat Wolfgang. The linux kernel configurator as feature modelling tool.

[TH03] Jean-Christophe Trigaux and Patrick Heymans. Software product lines: State of the art. Technical report, FUNDP, Namur, 2003.

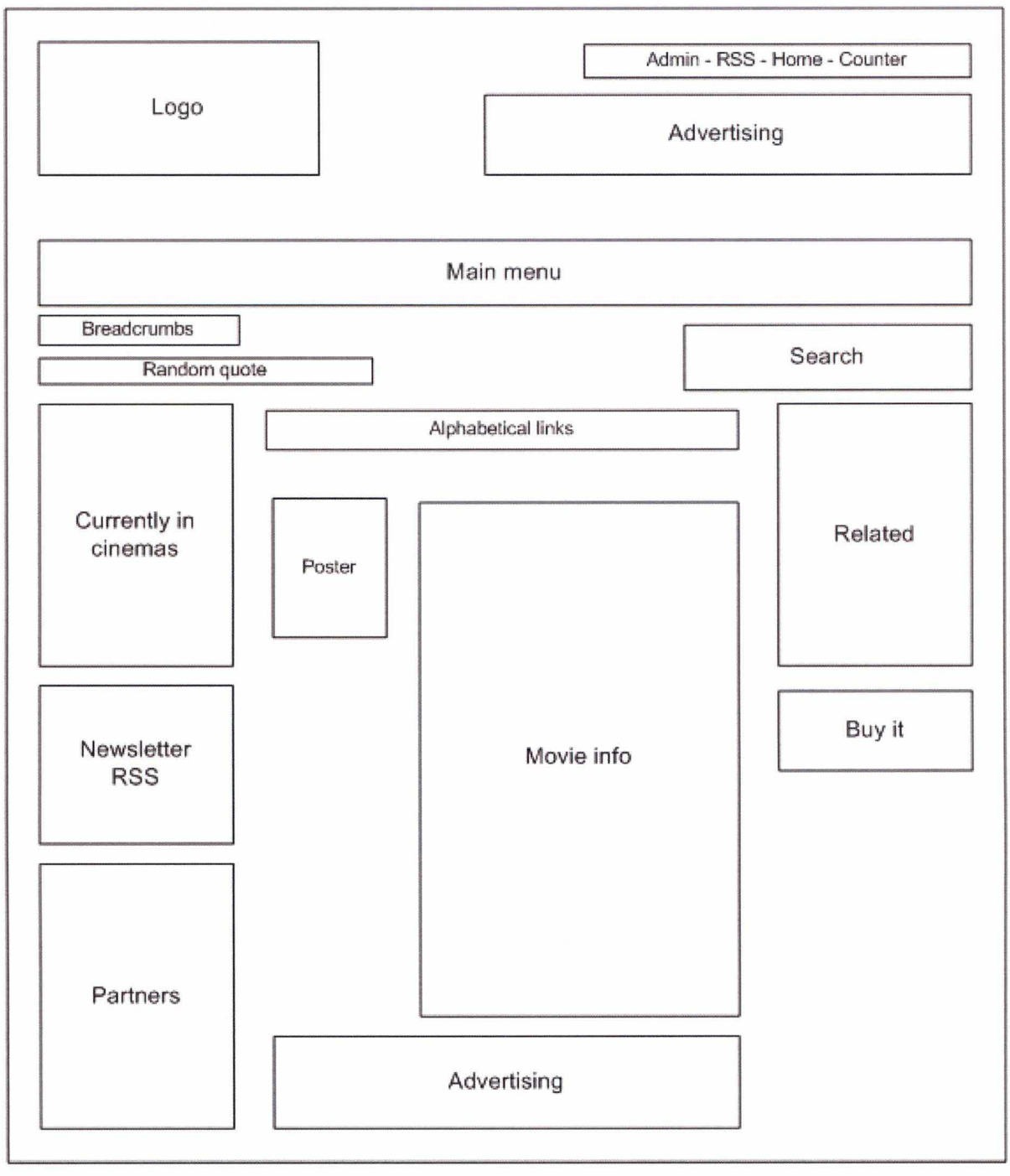
[VK00] Tuomo Vehkomäki and Kari Känsälä. A comparison of software product family process frameworks. In *IW-SAPF-3: Proceedings of the International Workshop on Software Architectures for Product Families*, pages 135–145, London, UK, 2000. Springer-Verlag.

[vL01] A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 249–262, 2001. Available from World Wide Web: `http://dx.doi.org/10.1109/ISRE.2001.948567`.

Appendix A

Sketches of websites

A.1 Film Critik



A.2 Actors

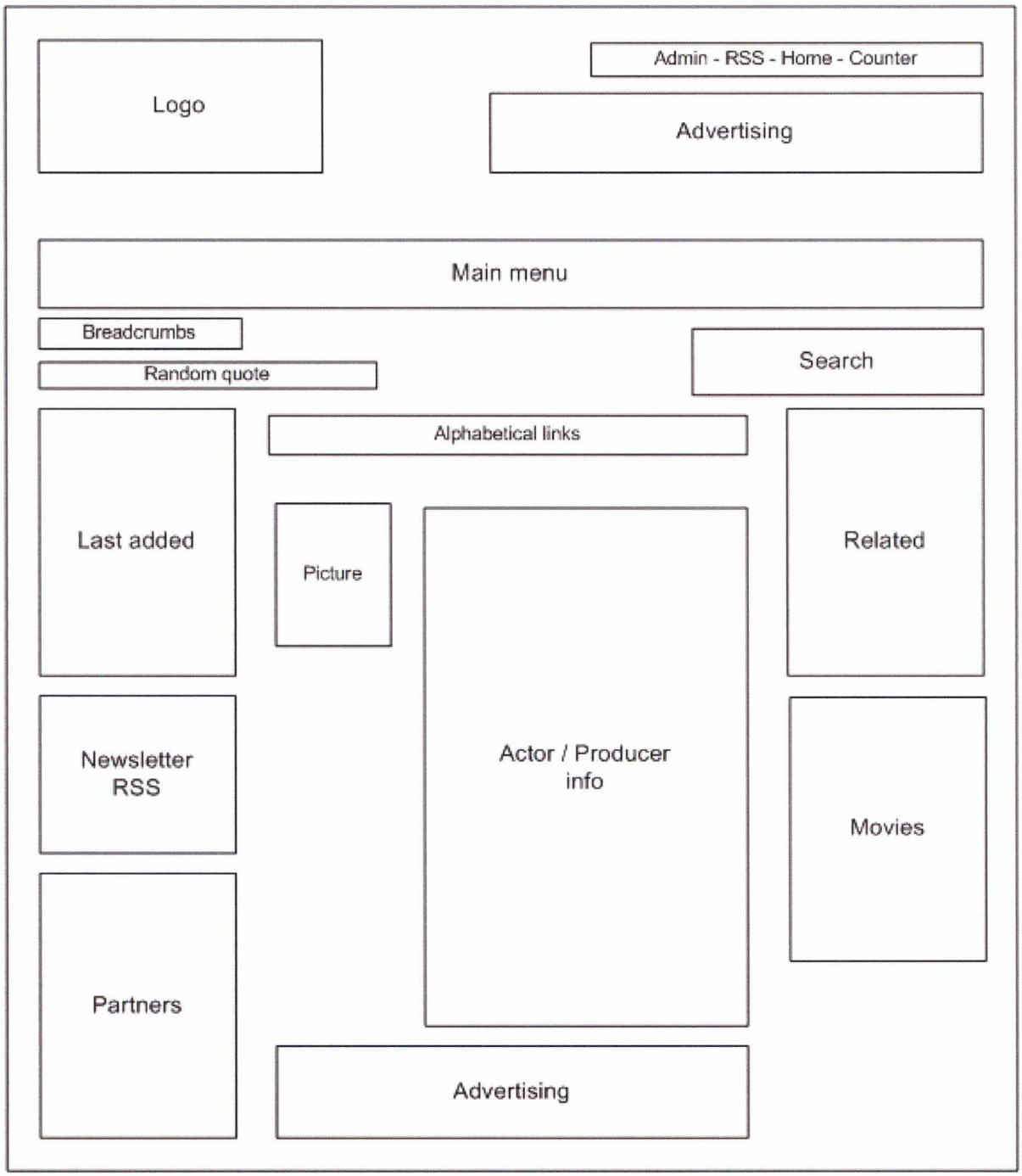


Figure A.2: Actor

A.3 Books

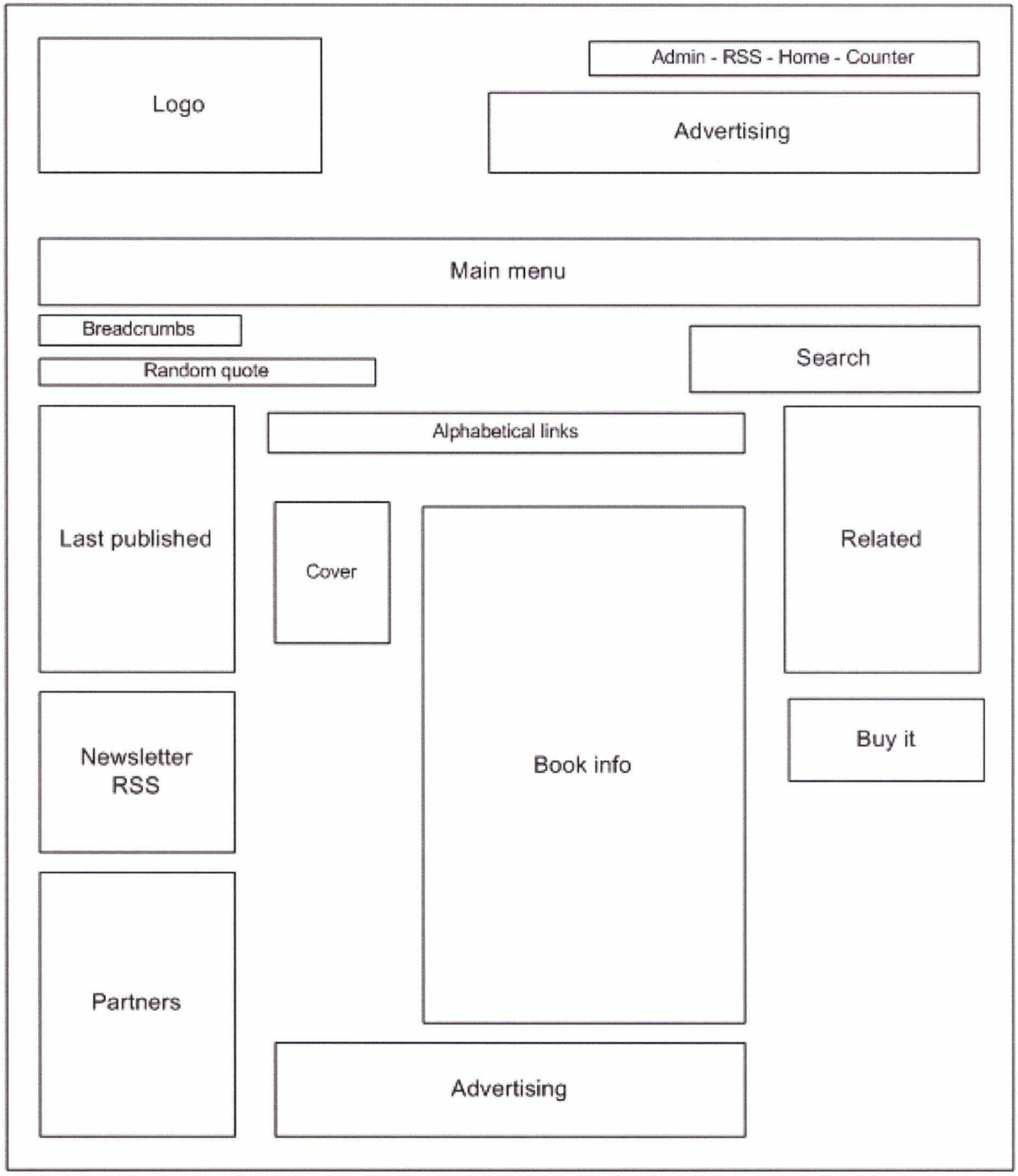


Figure A.3: Book review

A.4 Authors

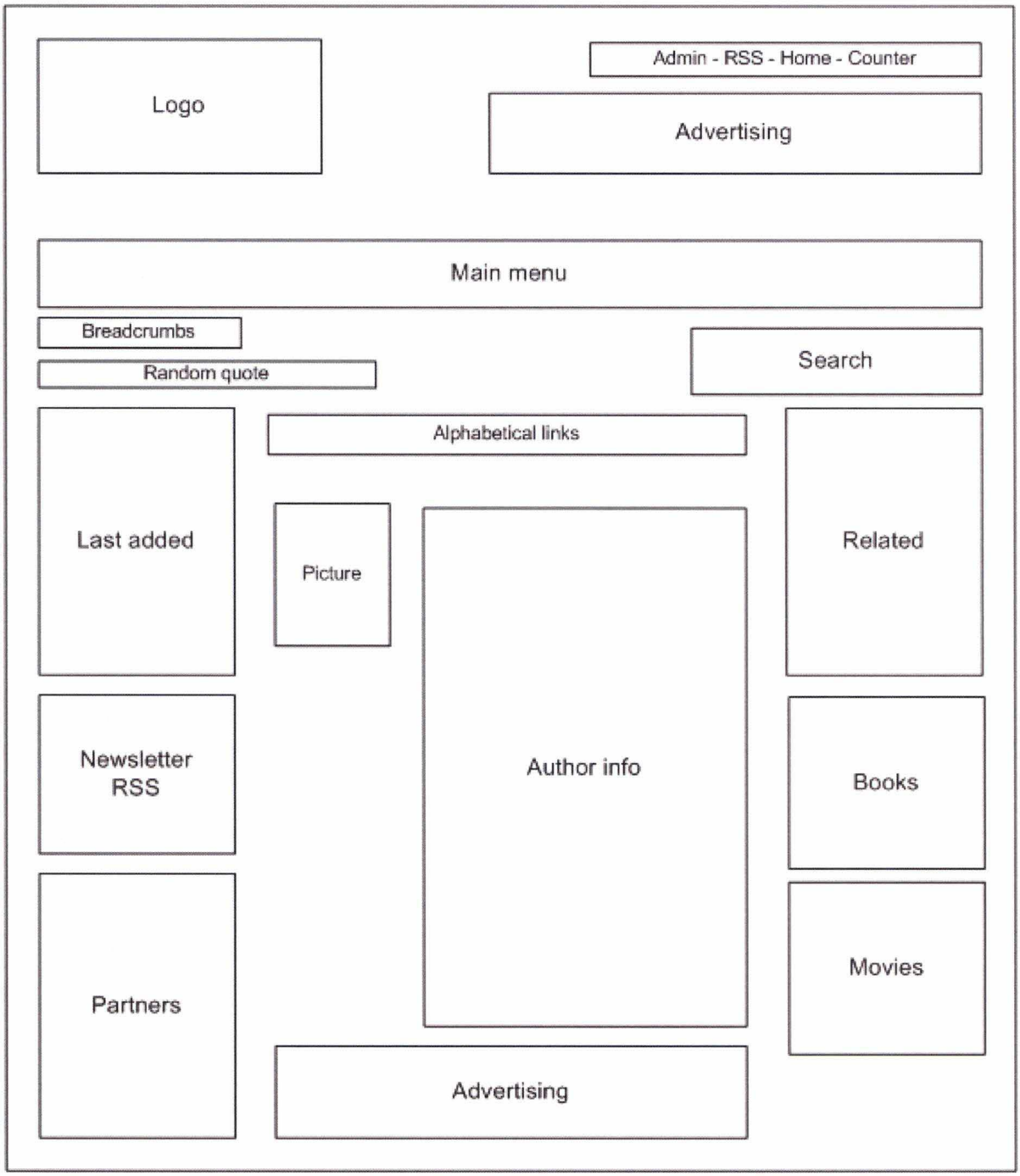


Figure A.4: Author

A.5 Soundtracks

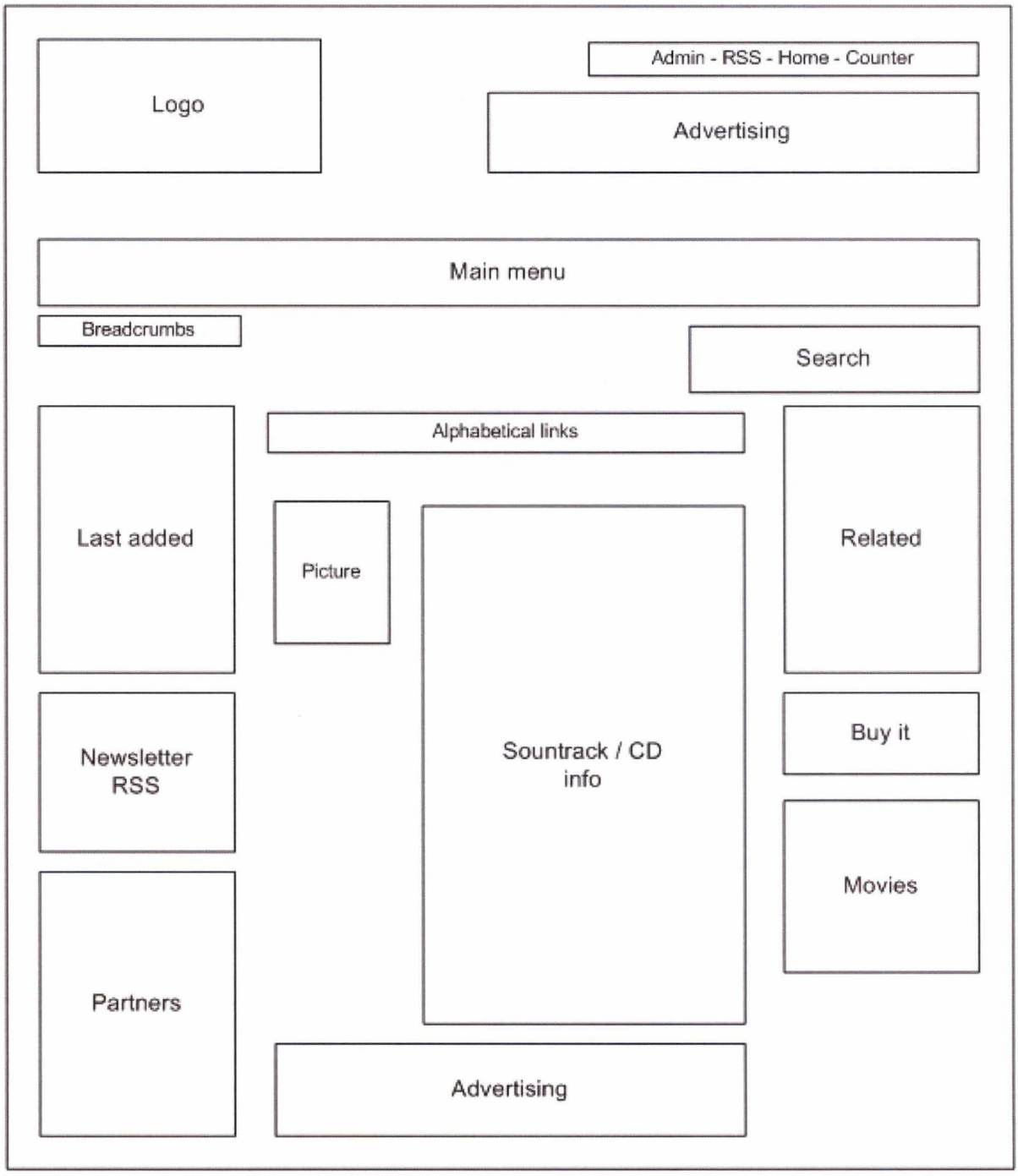


Figure A.5: Soundtrack

A.6 Musicians

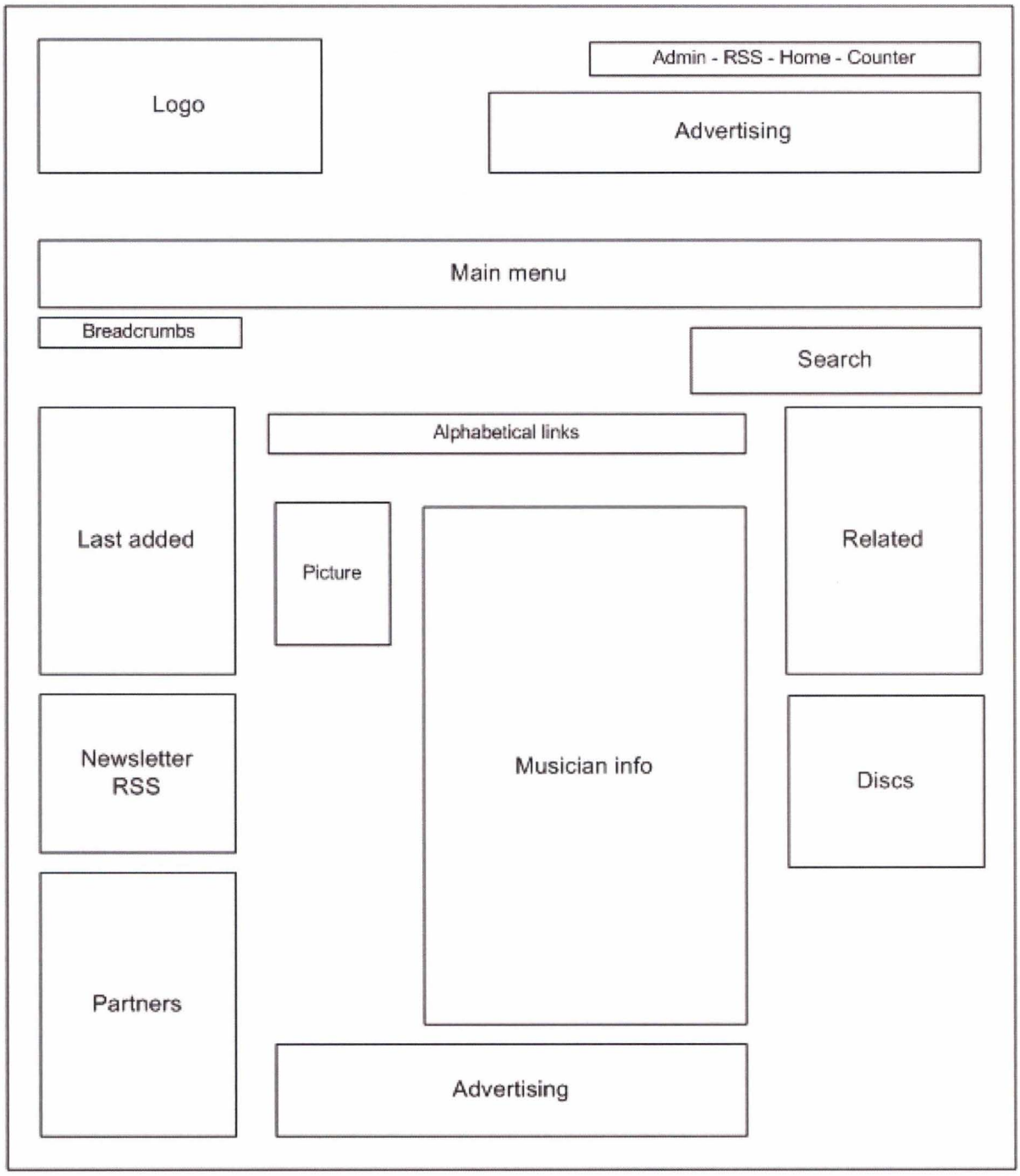


Figure A.6: Musician